**VŠB - Technická univerzita Ostrava**
**Fakulta elektrotechniky a informatiky**

# DIPLOMOVÁ PRÁCE

2015 Bc. Tomáš Hejret

**VŠB - Technická univerzita Ostrava**
**Fakulta elektrotechniky a informatiky**

# Rozhodovací stromy a jejich aplikace při analýze dat
# Decision Trees and Their Application in Data Analysis

2015 Bc. Tomáš Hejret

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Zadání diplomové práce

Student: **Bc. Tomáš Hejret**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Rozhodovací stromy a jejich aplikace při analýze dat
Decision Trees and Their Application in Data Analysis

Zásady pro vypracování:

Cílem práce je naimplementovat rozhodovací stromy ve vybraném programovacím jazyce a následně použít tuto implementaci při analýze dat (klasifikaci, regresi). Experimenty by měly probíhat nad reálnou kolekcí dat (předpokládá se využití některé z kolekcí z databáze UCI).

Body zadání:
1. Seznámení se s metodami pro klasifikaci a regresi s podrobnějším zaměřením na rozhodovací stromy.
2. Návrh a implementace rozhodovacích stromů ve vybraném jazyce.
3. Vytvoření sady experimentů a provedení srovnání s jinou vybranou metod pro klasifikaci.
4. Vymezení vhodných úloh a stanovení limitů použití aplikace.
5. Zhodnocení dosažených výsledků.

Seznam doporučené odborné literatury:

[1] Saeid Sanei, J. A. Chambers: EEG Signal Processing, Wiley-Interscience; 1 edition (September 11, 2007), ISBN-13: 978-0470025819

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Petr Gajdoš, Ph.D.**

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2015


doc. Dr. Ing. Eduard Sojka
*vedoucí katedry*

prof. RNDr. Václav Snášel, CSc.
*děkan fakulty*

## Poděkování

Tímto velice děkuji panu Ing. Petrovi Gajdošovi Ph.D. za odborné vedení a cenné rady při tvorbě této diplomové práce. Dále děkuji přítelkyni Ivě, rodině a přátelům za morální podporu a Michaele Vaňkové za jazykovou korekturu.

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 7.5.2015
Tomáš Hejret

## Abstrakt

Tato diplomová práce se zabývá metodami strojového učení, jejich srovnáním a detailněji pak metodou rozhodovacích stromů. V návaznosti zde lze nalézt metodiky určování kvality implementace metod strojového učení. Dále obsahuje popis konkrétní implementace metody rozhodovacích stromů, která je součástí této práce, experimentů s reálnými daty, provedených nad touto implementací a zhodnocení výkonnosti dle uznávaných metodik měření. V závěru práce se nachází výčet technik, které by realizovanou implementaci z různých hledisek vylepšily.

## Abstract

This master thesis discuses about machine learning methods, comparison of them and more in detail about decision trees method. Related to these methods there are listed methodologies of determination quality of machine learning methods. Thesis also includes description of particular decision trees implementation, which stands as a part of the thesis, experiments with real-life data, performed with the current implementation and evaluated performance according to relevant measure methodologies. At the end of the thesis there is a list of techniques, which could enhance particular implementation in various points of view.

## Klíčová slova

analýza dat, dolování dat, náhodné lesy, rozhodovací strom, strojové učení, učení s učitelem, klasifikace, regrese

## Key Words

data analysis, data mining, random forests, decision tree, machine learning, supervised learning, classification, regression

# Obsah

# 1 Úvod

Tato práce se nejprve zabývá seznámením se základními fakty o strojovém učení, jelikož jejím hlavním zaměřením je jedna z takovýchto metod. Dále je zde popsáno několik metod strojového učení, spolu s metodou, na kterou je práce zaměřena primárně, a také srovnání mezi nimi. Hlavní část práce pak představuje implementaci, jenž je její nedílnou součástí. V dalších částech práce lze nalézt popis konkrétní realizované implementace a její měření, testování a experimentování při použití reálných date z knihovny UCI. Výsledky experimentů a měření na těchto datech pak poslouží k vymezení vhodných úloh a limitů realizované implementace a návrhů na vylepšení stávajícího řešení.

Předpokládá se, že implementace dokáže generovat prediktivní modely pro regresi a klasifikaci dat a to pro spojitá, binární i kategorická data. Dalším předpokladem je možnost volby parametrů učení přes konfigurační soubor. Požadované nároky na výkonnost aplikace jsou kladeny zejména na proces zpracování dat (tedy vyhodnocování dat vygenerovaným prediktivním modelem) a to především na procentuální úspěšnost vyhodnocování a na časovou náročnost. Na procesu učení bude pozorována hlavně časová náročnost a také prostorová náročnost jak samotného procesu, tak výsledného prediktivního modelu.

### Motivace

Představme si rozsáhlou kolekci vícerozměrných dat (řádově desítky až tisíce) nebo datový tok (např. data z EEG, kamery) vícerozměrných dat. Abychom mohli daná data využít, potřebujeme z nich získat relevantní informace – musejí tedy projít jistou analýzou a zpracováním.

Analýzu a zpracování dat lze provést buď ručně, nicméně v případech, kdy jsou data velmi rozsáhlá (velká kolekce či velká dimenze dat) nebo je jejich tok v reálném čase velmi rychlý, nebude v lidských silách efektivně nalézat a rozpoznávat specifické události či entity, případně bude příliš náročný na lidské zdroje a jejich neustálou koncentraci, motivaci a další faktory, které snižují efektivitu lidských zdrojů. Na řadu proto přichází počítače a strojové učení, které tuto práci za člověka může udělat s mnohem vyšší efektivitou.

Další situací, kdy je potřeba, aby se stroj „rozhodoval“ sám, je absence člověka. Jedná se například o robotická vozítka na Marsu nebo jiné sondy vyslané do vesmíru, které člověk nemůže ovládat v reálném čase kvůli obrovskému zpoždění případných řídících signálů (způsobené velkou vzdáleností od zařízení).

Strojové učení se uplatňuje také v situacích, kdy by tradiční způsob programování, založený na ručně napsané soustavě podmínek, byl příliš složitý a také při hledání nových struktur, které jsou pro člověka zatím neznámé. Příkladem využití v tomto ohledu je dolování dat („data mining“). [8][18]

Celý tento motivační úvod by se dal shrnout a interpretovat také citátem od Johna Naisbitta: „*Topíme se v informacích, ale hladovíme po vědění.*“

# 2 Seznámení se strojovým učením

Jedná se o podoblast umělé inteligence, která se zabývá algoritmy umožňující počítačovému systému učit se. Obecně můžeme „učení se“ označit jako proces, při kterém si entita (hovoříme-li o strojovém učení, pak je touto entitou počítač) dokáže podle předané kolekce dat (tzv. „trénovací data“ nebo též „učicí data“) o entitách s určitou sadou atributů (či prediktorů) $A$, osvojit způsob, jakým předpovídat (odhadovat), třídit do skupin (tzv. klasifikovat) nebo shlukovat na základě podobnosti jiné entity dle dat (tzv. „testovací data“) o daných entitách se stejnou či velmi podobnou sadou atributů $A'$ o jiných entitách.

Zde se hodí uvést jeden výstižný citát Ryszarda Michalskeho, který v podstatě vyjadřuje obsah odstavce výše: „*Učení je konstruování nebo upravování reprezentací toho, co bylo zažito.*“

## 2.1 Kritéria dělení metod strojového učení

Strojové učení lze rozdělit podle několika kritérií:

- Přítomnost učitele při učení:
  - bez učitele,
  - s učitelem,
  - kombinované – posilované učení (tzv. „reinforcement learning“) - část trénovacích dat je se známým výstupem a obvykle větší část dat je bez něj).
- Způsob učení:
  - dávkové (všechna trénovací data jsou potřeba před začátkem učení),
  - inkrementální (naučený model může upravit vlastní strukturu, aby byl schopen rozpoznávat i nové typy entit).
- Typ atributů:
  - binární,
  - kategoriální,
  - celočíselné,
  - reálné (spojité).
- Zaměření (typ úlohy):
  - klasifikace – přiřazování tříd objektům,
  - regrese – odhad číselné hodnoty podle vstupu,

- shlukování (v podstatě jde o klasifikaci po učení bez učitele) – objekty, vykazující podobné vlastností, se shlukují.

- „White box“ (člověk je schopen pochopit vnitřní fungování a také je možné model interpretovat např. soustavou booleanovských pravidel) / „back box“ model.
- Parametrická metoda (založena na předpokladech o vlastnostech entit) / neparametrická metoda.

## 2.2 Příklady využití strojového učení v praxi

Dnešní doba by se dala bez nadsázky označit jako éra výpočetních a komunikačních technologií. Postupem času se elektronika dostala prakticky všude, a i běžné věci se tak stávají „chytrými“. Někdy se za touto chytrostí skrývá jen aplikace v chytrém telefonu nebo jiné spotřební elektronice, jejíž autor měl dobrý a inovativní nápad, ale implementace nevyžadovala učící se algoritmus.

Nasazení strojového učení má velmi široké uplatnění. S mnoha aplikacemi strojového učení se běžně setkáváme každý den a možná o tom ani nevíme. Nyní si ukažme několik příkladů, kde všude se lze v běžném životě setkat se strojovým učením a u kterých často bývá použitou metodou právě metoda rozhodovacích stromů (nebo její modifikace), které je tato práce věnována především:

- detekce objektů v obraze - např. detekce obličeje / úsměvu při fotografování (např. fotografický software pro fotoaparáty a chytré telefony, Kinect),
- rozpoznávání mluvené řeči,
- nabídka souvisejících či podobných článků na webových stránkách nebo doporučení zboží na základě obsahu nákupního košíku nebo již prohlédnutého zboží v internetovém obchodě,
- cílená reklama,
- detekce spamu,
- a další.

Také existuje celá řada implementací knihoven či nástrojů pro analýzu dat Vybírám ty, které implementují některou z variant rozhodovacích stromů, případně i další metody, např.:

- knihovna Sherwood, https://github.com/flyfj/MachineLearning/tree/master/msrc_random_forest/csharp/lib
- knihovna Random Forests (Leo Breiman a Adele Cutler), http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
- knihovna Orange pro Python, http://orange.biolab.si

- knihovna VFML (Very Fast Machine Learning) v jazyce C, http://www.cs.washington.edu/dm/vfml
- open source nástroj WEKA (Waikato Environment for Knowledge Analysis), https://weka.wikispaces.com/
- a další.

# 3 Metody strojového učení

Každá z metod najde své uplatnění pro jiný typ dat. Některé metody mohou být univerzálnější, ale na druhou stranu nemusí poskytovat takový výkon pro některé typy dat, jako to dokáží metody jiné. Některé z nich zase nedokáží zpracovávat všechny typy dat (např. mohou zpracovávat pouze data kategoriální, ale neporadí si s číselnými daty). Metody se liší také odolností vůči šumu – tedy hodnotám vymykajícím se standardu. Žádnou z metod tedy nelze označit za jednoznačně nejlepší, nejpřesnější nebo nejvýkonnější a vždy je nutné zvážit pro jaký účel je strojové učení určeno a s jakými daty má pracovat. [1]

Tato kapitola se bude nepříliš do hloubky zabývat několika metodami strojového učení včetně metody rozhodovacích stromů, jež bude rozebrána trochu podrobněji, jelikož je tato diplomová práce zaměřena především na ni. Nenaleznete zde kompletní seznam metod ani jejich detailní popis, který by čtenáři zaručil, že je bude schopen komplexně pochopit a implementovat – spíše jde o účel srovnání vlastností metod s metodou rozhodovacích stromů.

## 3.1 Neuronové sítě (Artificial Neural Networks - ANNs)

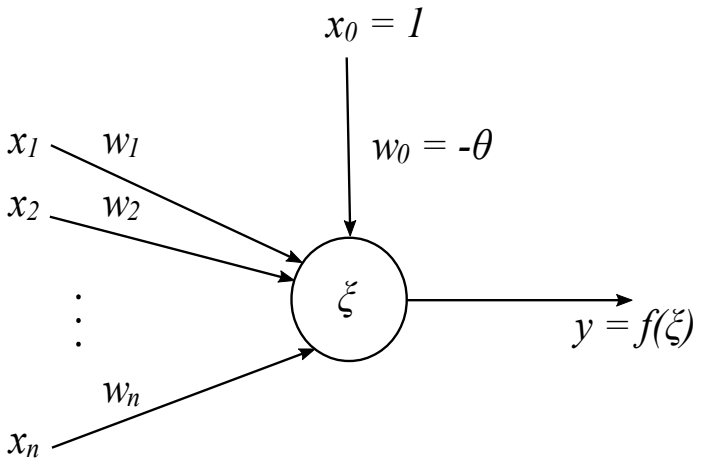
Jedná se o statistickou, neparametrickou, biologicky inspirovanou metodu, která byla navržena zejména pro strojového učení, rozpoznávání vzorů („pattern recognition“) a analýzu více proměnných. Jedná se o „black box“ model, který zvládne zpracovávat kategorická i spojitá data. [2][8]

### Základní princip

V neuronové síti jsou základními stavebními prvky neurony, podobně jako v případě lidského mozku, kterým se metoda inspiruje. Do každého neuronu ($\xi$) vede libovolný počet vstupů ($x_i$) ohodnocených různými vahami ($w_i$), dále jeden vstup s konstantní prahovou hodnotou ($\theta$), který není připojen k žádnému jinému neuronu a který určuje, kdy se neuron aktivuje, a pouze jediný výstup $f(\xi)$, jež je klasifikací či numerickou hodnotou regrese. *Neurony jsou vzájemně propojeny a navzájem si předávají signály a transformují je pomocí určitých přenosových pravidel.* [2]

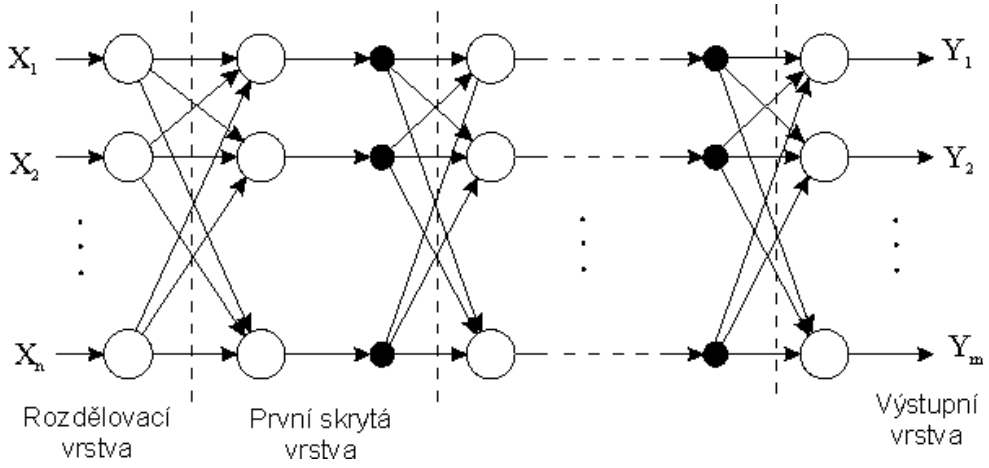
Obecný model neuronu s libovolným počtem vstupů $n$ lze vidět na Obrázek 1 a jeho obecnou transformační funkci $f$ můžeme popsat vzorcem vytvořeným neurovědcem McCullochem a logikem Pittsem:

$$y = f\left(\sum_{i=1}^{n} x_i \cdot w_i - \theta\right)$$




**Obrázek 1 - Obecný model neuronu** [7]

Samotné neurony toho moc nesvedou a jejich hlavní síla se projeví až po vytvoření sítě neuronů. Neurony bývají sdružovány obvykle do vrstev. Celý algoritmus se učí sám včetně volby vah. Příklad jednoduché vícevrstvé neuronové sítě lze vidět na Obrázek 2.




**Obrázek 2 - Příklad vícevrstvé neuronové sítě** [2]

Rozdělovací nebo také vstupní vrstva přijímá hodnoty ke zpracování a přivádí je na vstupy všech neuronů následující vrstvy. Počet skrytých (nebo také vnitřních) vrstev se odvíjí od složitosti funkce, kterou má síť představovat. Poslední vrstvou je vrstva výstupní, jejíž výstupní hodnoty představují odezvu celé sítě. [2][7][8]

## Výhody:

- robustnost,
- lze použít jak pro klasifikaci, tak pro regresi,

- učení může probíhat s učitelem, bez učitele, nebo posilováním.

**Nevýhody:**

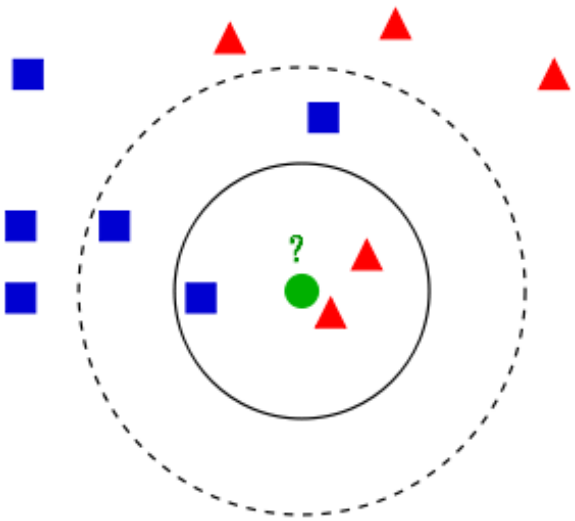
- hrozba přeučení sítě (tzv. „overfitting“),
- složité nastavení a testování parametrů,
- prediktivní model má charakter „black box“.

## 3.2 Metoda k-nejbližších sousedů (k-Nearest Neighbors - k-NN)

Tato neparametrická metoda bývá nejčastěji používána pro klasifikaci, ale lze ji použít také pro regresi, při které ale nedosahuje tak dobrých výsledků. Model má charakter „white box“.

### Základní princip

Základní princip vychází z předpokladu, že data, která chceme zpracovat, budou mít podobné vlastnosti (hodnoty atributů), jako některé datové body z trénovací množiny dat, u kterých známe zařazení do správné třídy (klasifikace), resp. hodnotu jeho cílové proměnné (regrese). Při zpracování vstupního datového bodu se tento zobrazí do stejného prostoru jako je prostor trénovací množiny dat, následně se výpočtem vzdáleností od všech trénovacích datových bodů nalezne *k* nejbližších sousedů (znázorněno na Obrázek 3) a datový bod je pak zařazen do třídy, do které patří nejvíce z nejbližších sousedů (klasifikace), resp. mu je přidělen průměr cílových hodnot nejbližších sousedů (regrese). Pro výpočet vzdáleností mezi datovými body se nejčastěji používá euklidovská vzdálenost, případně Hammingova metrika. [2][9]




**Obrázek 3 - Znázornění zobrazení zpracovávaného datového bodu do prostoru trénovacích dat s jeho nejbližšími sousedy** [9]

**Výhody:**

- snadná implementace,
- nízké výpočetní nároky,
- vysoká účinnost,
- model má charakter „white box“.

**Nevýhody:**

- méně vhodná pro regresi,
- vhodná spíše pro menší množství tříd,
- menší stabilita.

## 3.3 Diskriminační analýza (Discriminant Analysis - DA)

Jde o metodu lineárního modelování, která se používá jak ke klasifikaci a rozpoznávání vzorů, tak pro hledání lineárních kombinací atributů, které charakterizují společné znaky (atributy) dat ze stejné třídy a odlišující od dat z odlišných tříd. Tato metoda je úzce spjata se statistickou metodou ANOVA („analysis of variance“), regresní analýzou a dalšími metodami, jenž mají stejné uplatnění, ale jsou určeny pro odlišné typy dat.

### Základní princip

DA probíhá v podstatě ve dvou krocích. Prvním je testování rozdílů mezi vysvětlujícími atributy (deskriptory) v předem definovaných skupinách. Máme-li statisticky významné rozdíly mezi skupinami M atributů, pak ve druhém kroku přejde analýza ke hledání lineární kombinace (tzv. *„diskriminační nebo identifikační funkce“* [2]) z M atributů, která nejlépe diskriminuje tyto skupiny. K vlastní klasifikaci je využita vzdálenost („score“) měřící spolehlivost klasifikace a představující vzdálenost od přímky oddělující roviny. K výpočtu této vzdálenosti se užívá diskriminačních rovnic:

$$f_{ik} = \sum_{p} (x_{ip} - \bar{x}_p) c_{pk}$$

$f_{ik}$ – jednotlivé vzorky

$c_{pk}$ – normalizované složky vlastních vektorů

$x_{ip}$ - diskriminační skóre

Testovaná data se rozdělují do tříd s určitou pravděpodobností na základě výsledků diskriminačních rovnic. [2][10]

**Nevýhody:**

- přítomnost odlehlých hodnot není přípustná,
- předpokládá se normální rozložení v rámci kategorií,
- jsou potřeba větší sady dat.

## 3.4 Logistická regrese (Logistic Regression – LR)

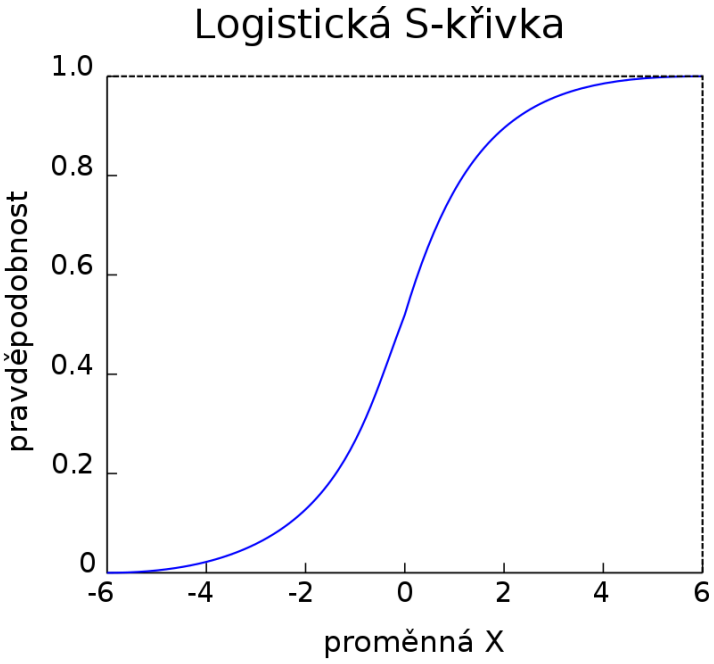
Metoda logistické regrese je použitelná pro klasifikaci pouze do dvou tříd – tedy zda zkoumaný jev nastal či nenastal. Metoda v podstatě provádí odhad pravděpodobnosti výskytu určitého jevu na základě atributů známých dat.

### Základní princip

Jak již bylo uvedeno, touto metodou klasifikujeme do dvou tříd – 1 (pokud jev nastal, resp. pravděpodobnost, že jev nastal, je větší než 0,5) nebo 0 (jev nenastal, resp. pravděpodobnost, že jev nastal, je menší než 0,5). K vytvoření dané podmínky potřebujeme transformační funkci:

$$g(p)=\ln\frac{p}{1-p}$$

Závislost má tvar S-křivky (viz. Obrázek 4).




**Obrázek 4 - S-křivka logistické regrese** [2]

Logistický regresní model pravděpodobnosti příslušnosti bodu do třídy pro $p$ nezávisle proměnných a s regresními koeficienty $\beta$ může vypadat následovně:

$$\pi(x) = P(Y = \frac{1}{x}) = \frac{1}{1 + e^{\sum_p \beta_p \cdot x_p}}$$

což v kombinaci s předchozí rovnicí vychází:

$$g(x) = \ln\left(\frac{P(Y=1)}{1 - P(Y=1)}\right) = \sum_p \beta_p \cdot x_p$$

g(x) zde představuje lineární regresní model.

Parametry modelu se odhadují metodou maximální věrohodnosti. Za nejvíce věrohodný je považován odhad, který maximalizuje L:

$$\ln L = \sum_{i=1}^{n} \left(y_i \cdot \ln(\pi(x)) + (1 - y_i) \cdot \ln(1 - \pi(x))\right)$$

Model je nelineární a tak je odhad parametrů složitější. S výhodou se zde používají různé iterační techniky.

Máme-li odhady regresních parametrů, je nutné zjistit jejich významnost, což provedeme testováním nulové hypotézy:

$$H_0 : \beta = \beta^0$$

pro

$$\beta^0 = (\beta_0, \beta_1, \ldots, \beta_p)$$

Nyní provedeme test významnosti parametrů, pro což existuje několik vhodných postupů – více se lze o nich dočíst ve zdrojích této práce. [2][11][12]

### Výhody:

- atributy mohou být kombinací kategorických a spojitých hodnot.

### Nevýhody:

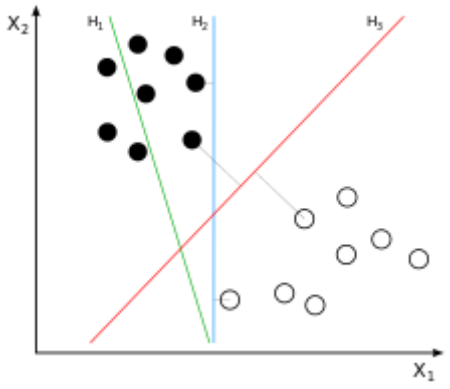
- klasifikace pouze do dvou tříd,
- předpoklad binomického rozložení dat,
- netoleruje odlehlé hodnoty.

## 3.5 Algoritmy podpůrných vektorů (Support vector machines – SVM)

Tato semiparametrická metoda představuje strojové učení s učitelem, případně v některých modifikacích také kombinované. S její pomocí lze hledat vzory ve spojitých datech, díky kterým lze provádět klasfikaci i regresní analýzu.

### Základní princip

Princip metody spočívá v hledání nadroviny, která od sebe oddělí jednotlivé třídy v trénovacích datech (viz. Obrázek 5). V ukázce na obrázku vidíme 3 příklady nadrovin, rozdělující prostor. Nadrovina $H_1$ nerozděluje třídy, její použití by proto bylo zcela nevhodné. Nadrovina $H_2$ sice od sebe zcela rozděluje třídy, ale vzdálenosti nejbližších datových bodů jsou velmi malé. Nejlepším řešení se tedy jeví nadrovina $H_3$.




**Obrázek 5 - Příklady možností separace datových bodů jednotlivých tříd nadrovinami** [13]

Je zde patrná souvislost s metodou rozhodovacích stromů, u kterých ale prostor dělíme pouze „horizontálně“ či „vertikálně“. [2][13]

### Výhody:

- data nemusí splňovat požadavky na rozložení,
- vysoká stabilita,
- odolnost vůči šumu.

### Nevýhody:

- pouze pro spojitá data,
- výsledný model je typu „black box“.

## 3.6 Rozhodovací stromy (Decision Trees - DT)

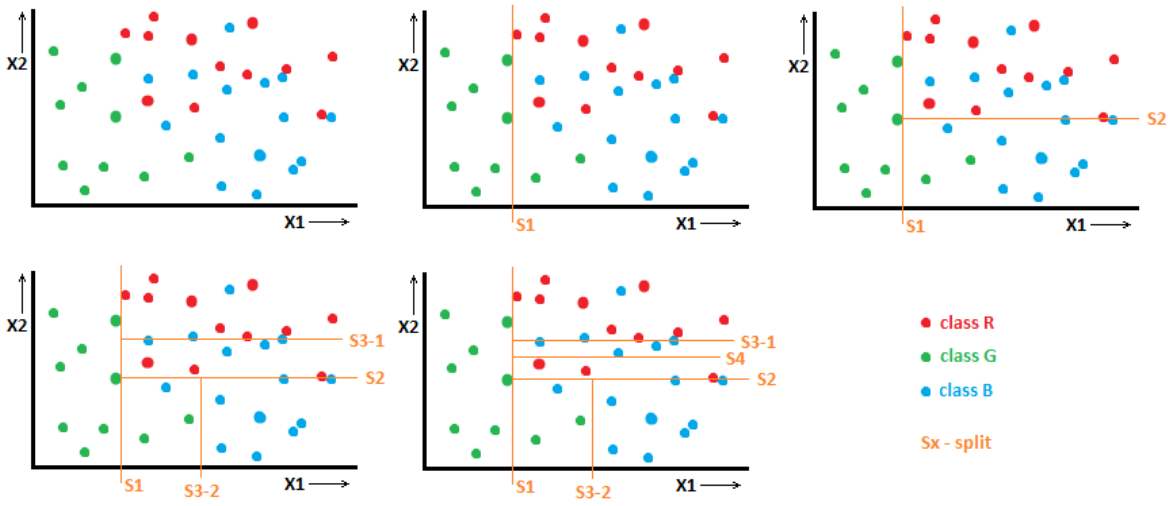
Tato statistická metoda umožňuje zpracování velkých rozsahů smíšených (binárních, kategoriálních, celočíselných i spojitých) dat a jejich přidělování do tříd s co možná největší separací. Výhodou rozhodovacích stromů je, že soubor trénovacích dat nemusí splňovat požadavky jako je např. Gaussovo rozdělení. [2]

Rozhodovací stromy můžeme dle typů dělení z kapitoly 2.1 zařadit jako učení s učitelem nebo bez učitele (dle propracovanosti konkrétní implementace metody); učení probíhá dávkově; metoda si může poradit se všemi vyjmenovanými typy atributů; stejně tak úlohy, se kterými si rozhodovací stromy umí poradit mohou být všechny jmenované (opět dle propracovanosti) a výsledný model je typu „white box“. [3]

### Základní princip

Princip fungování této metody spočívá v dělení trénovacího prostoru instancí tříd (klasifikace) na podprostory, ve kterých se dělením snižuje míra neuspořádanosti. Prostor se rozdělí na tolik podprostorů, aby v každém takovém podprostoru byly zastoupeny v ideálním případě pouze instance stejné třídy (entropie = 0), reálně pak alespoň výrazně většinově zastoupeny instance jedné třídy. Když se dělením prostoru dostaneme do situace, že obsahuje pouze instance jedné třídy nebo jsou tyto výrazně většinově zastoupeny (entropie ≈ 0), pak se uvažuje, že daný podprostor obsahuje s vysokou pravděpodobností instance právě jedné konkrétní třídy. Příklad dělení dvourozměrného prostoru lze vidět na Obrázek 6. Rozdělení prostoru podmínkou X1 < S1 získáme 2 podprostory, z nichž jeden podprostor čeká další dělení na podprostory a druhý obsahuje instance pouze jediné třídy (třídy G). A to je právě stav, kterého se snažíme v ideálním případě dosáhnout.

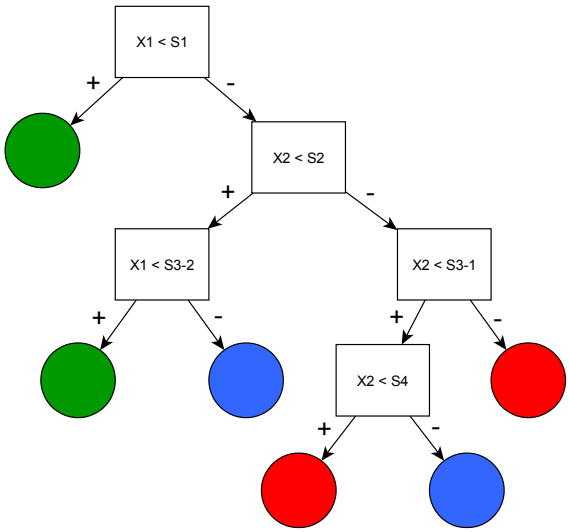
Výpočet, pomocí kterého se hledá nejvhodnější místo (rozměr a hodnota) pro dělení intervalu, se nalézá níže v této kapitole.




**Obrázek 6 - Dělení prostoru na podprostory dle spojité proměnné**

Implementace, kterou je tohoto dělení prostoru dosaženo, tvoří rozhodovací strom. Rozhodovací strom sestává z kořene, který se následně dělí na větve dle jedné podmínky (např. $x > 3.8$) v případě číselných dat. V případě kategoriálních dat se dělí na N větví (s N podmínkami), kde N představuje počet přípustných hodnot kategoriálního atributu. Dále se dělí také větve a v bodě, kdy se v daném podprostoru vyskytují pouze (nebo převážně) třídy jednoho typu, vznikne list. List představuje konečnou fázi v budování větve stromu. Strom lze chápat jako orientovaný graf.

Obrázek 7 demonstruje, jak by vypadal rozhodovací strom rozdělující prostor tak, jak je uvedeno na Obrázek 6.




**Obrázek 7 - Příklad rozhodovacího stromu pro číselné proměnné**

U binárních a číselných atributů se uzly stromu dělí na právě 2 větve, zatímco u kategoriálních atributů má uzel právě tolik větví, kolik je možných validních hodnot daného kategoriálního atributu. Uvedu zde tedy také příklad (převzatý a přeložený [16]) s kombinací binárních a kategoriálních dat.

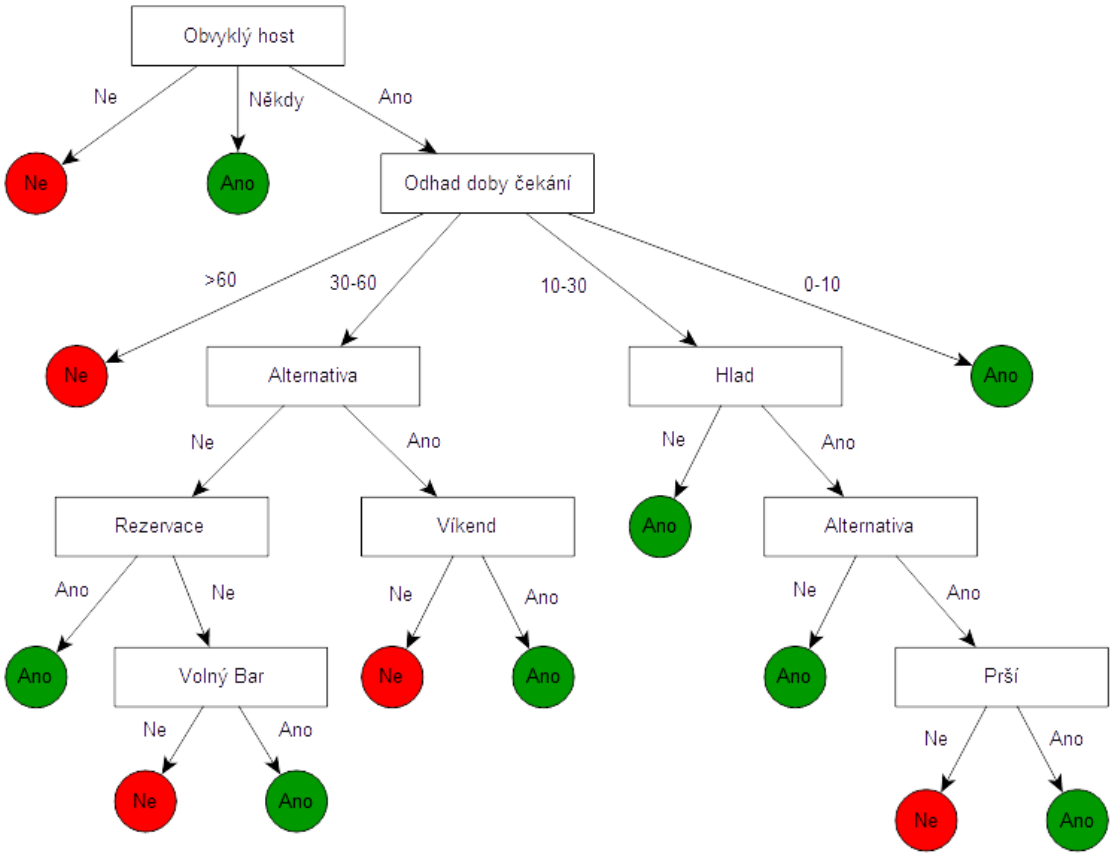
Nejprve si nastíníme problém tabulkou Tabulka 1, který představuje vzorový případ chování zákazníků restaurace. Máme zde 12 případů (zákazníků), kteří se rozhodují podle několika kritérií (atributů), zda počkají nebo nepočkají (výstupní třída je tedy stav rozhodnutí hosta) na volný stůl v této restauraci.

| Případ | Atributy | | | | | | | | Třída |
|---|---|---|---|---|---|---|---|---|---|
| | Alternativa | Volný Bar | Víkend | Hlad | Obvyklý host | Prší | Rezervace | Odhad doby čekání | Bude čekat |
| X1 | Ano | Ne | Ne | Ano | Někdy | Ne | Ano | 0-10 | Ano |
| X2 | Ano | Ne | Ne | Ano | Ano | Ne | Ne | 30-60 | Ne |
| X3 | Ne | Ano | Ne | Ne | Někdy | Ne | Ne | 0-10 | Ano |
| X4 | Ano | Ne | Ano | Ano | Ano | Ne | Ne | 10-30 | Ano |
| X5 | Ano | Ne | Ano | Ne | Ano | Ne | Ano | >60 | Ne |
| X6 | Ne | Ano | Ne | Ano | Někdy | Ano | Ano | 0-10 | Ano |
| X7 | Ne | Ano | Ne | Ne | Ne | Ano | Ne | 0-10 | Ne |
| X8 | Ne | Ne | Ne | Ano | Někdy | Ano | Ano | 0-10 | Ano |
| X9 | Ne | Ano | Ano | Ne | Ano | Ano | Ne | >60 | Ne |
| X10 | Ano | Ano | Ano | Ano | Ano | Ne | Ano | 10-30 | Ne |
| X11 | Ne | Ne | Ne | Ne | Ne | Ne | Ne | 0-10 | Ne |
| X12 | Ano | Ano | Ano | Ano | Ano | Ne | Ne | 30-60 | Ano |

**Tabulka 1 - Tabulka nastalých případů (trénovací data)**

Legenda:

- Alternativa – v blízkosti se nachází adekvátní „náhradní“ restaurace.
- Volný Bar – bar, na kterém by host mohl strávit čekání na volný stůl, je volný.
- Víkend – jedná se o víkendový den.
- Hlad – host je velmi hladový.
- Obvyklý host – host restaurace navštěvuje pravidelněji.
- Prší.
- Rezervace – host má stůl rezervován.
- Odhad doby čekání.
- Bude čekat (výstupní třída).




**Obrázek 8 - Příklad rozhodovacího stromu pro binární a kategoriální atributy**

Pro tato trénovací data by mohl rozhodovací strom vypadat například jako na Obrázek 8.

Jak lze vidět z příkladů vybudovaných rozhodovacích stromů, tomuto rozhodovacímu modelu může člověk bez problémů porozumět („white box“).

To, co bylo uvedeno v této kapitole, lze považovat za ideální stav. Reálné aplikace ale takto jednoduché nejsou. Z tohoto důvodu se do algoritmu přidávají různé metody, které mají za úkol vylepšit kvalitu výsledného rozhodovacího modelu. Tyto metody pro zvýšení použitelnosti na reálných datech lze nalézt v kapitole 5.1 .

Zásadním problém při budování stromu vzniká při určování, který atribut (a jakou hodnotu daného atributu - v případě spojitých atributů) zvolit, abychom vybudovali co možná nejoptimálnější strom, tedy strom s co nejmenším počtem uzlů. Každý vybudovaný uzel stromu s podmínkou („rozhodnutím“) zvyšuje prostorovou (zaujímá v paměti více místa) i časovou složitost (při vyhodnocování trénovacího či testovacího datového vektoru *v* představuje každý uzel „rozhodnutí“, které je potřeba vykonat pro zařazení vektoru *v* do třídy). Rozdílně se pracuje

s trénovacími daty pro účel klasifikace (máme konečné množství variant, pro které se může strom „rozhodnout“) oproti trénovacím datům, u kterých je účelem vymodelovat regresní model (výstupem je spojitá hodnota).

### Hledání nejvhodnějšího bodu pro dělení prostoru při klasifikaci

V případě klasifikačního úkolu má každý uzel stromu za úkol zvýšit homogenitu (z hlediska četnosti tříd) podprostorů, které svou podmínkou rozdělil z původního prostoru, neboli snížit v jednotlivých podprostorech entropii (míru neuspořádanosti) na co nejnižší hodnotu. Výpočet entropie (nebo také „nečistoty“ - impurity) nám tedy pomůže v hledání nejvhodnějšího atributu. Entropie se počítá pro každý z podprostorů zvlášť:

$$E = \sum_i - p_i \cdot \log_2 p_i$$

$p_i$ – pravděpodobnost výskytu třídy $i$

Z tohoto vyplývá, že minimální entropie představuje stav, kdy se v podprostoru nacházejí instance pouze jediné třídy.

Entropie samotná by byla pro posuzování kvality dělení prostoru poměrně nedostačující. Byla proto zavedena míra zvaná informační zisk („Information Gain“) $I_E$, udávající relativní hodnotu získaného množství informace a vztahuje se ke konkrétnímu místu $N$ dělení prostoru. Nejprve ale zavedeme váženou entropii $E_W$ potřebnou pro výpočet informačního zisku:

$$E_{Wx} = \frac{k_x}{k} \cdot E_x$$

index $x$ – představuje podprostor $x$

$k$ – počet datových bodů v rodičovském prostoru

$k_x$ – počet datových bodů v podprostoru

$$I_E(N) = E_p - (E_{Wa} + E_{Wb})$$

$E_p$ – entropie rodičovského prostoru, děleného na podprostory

$E_{Wa}$ – vážená entropie podprostoru $a$

$E_{Wb}$ – vážená entropie podprostoru $b$

### Hledání nejvhodnějšího bodu pro dělení prostoru při regresi

Má-li být úkolem prediktivního modelu regrese, pak prostor dělíme na podprostory na základě míry, tzv. „redukce odchylky“ („Variance Reduction“) $I_V$ vztažené ke konkrétnímu místu $N$ dělení prostoru. Tato míra udává, jak dobře byly od sebe odděleny datové body, které jsou od sebe

v prostoru více vzdáleny a výsledná hodnota výpočtu je přímo úměrná kvalitě rozdělení. Takové místo N, které bude mít nejvyšší redukci odchylky pak bude použito pro dělení prostoru. Výpočet se provádí dle vzorce:

$$I_V(N)=\frac{1}{|S|}\sum_{i\in S}\sum_{j\in S}\frac{1}{2}(x_i-x_j)^2-\left(\frac{1}{|S_t|}\sum_{i\in S_t}\sum_{j\in S_t}\frac{1}{2}(x_i-x_j)^2+\frac{1}{|S_f|}\sum_{i\in S_f}\sum_{j\in S_f}\frac{1}{2}(x_i-x_j)^2\right)$$

$x$ – cílová proměnná

$S$ – rodičovský prostor

$S_t$ – podprostor $t$ („true“), do kterého patří datové body, jež podmínku splnily

$S_f$ – podprostor $f$ („false“), do kterého patří datové body, jež podmínku nesplnily

V každé iteraci počítáme informační zisk $I_E$, respektive redukci odchylky $I_V$ pro každý atribut každého datového bodu v daném rodičovském prostoru jako možného kandidáta na bod dělení. Zvolíme ten atribut s příslušnou hodnotou, u kterého bylo dosaženo dělení s největším $I_E$, respektive s největší $I_V$. [2][3][18]

**Výhody**:

- nízké nároky na přípravu dat - není nutné, aby byla datová kolekce normalizována či aby byly vytvořeny umělé proměnné („dummy variables“),
- schopnost zpracovávat číselná i kategorická data,
- robustní,
- zvládá analyzovat v rozumném čase se standardními výpočetními zdroji rozsáhlé kolekce dat,
- možnost validovat model použitím statistických testů pro zjištění důvěryhodnosti modelu,
- použitelnost jak pro učení s učitelem, tak bez učitele.

**Nevýhody:**

- problém učení optimálního rozhodovacího stromu je znám jako NP-úplný,
- existují koncepty složité pro naučení rozhodovacích stromů a není snadné je pomocí rozhodovacích stromů namodelovat – např. nonekvivalence, multiplexerové problémy či parita,
- algoritmy pro učení stromu mají přirozeně tendenci k budování příliš detailního stromu (tzv. „overfitting“), což celý model degraduje, a je proto nutné se s ním vypořádat. Možnosti řešení tohoto problému naleznete v kapitole 5.1 ,
- pro data zahrnující kategoriální hodnoty s různými počty validních kategorií je informační zisk přikloněn k atributům s vyšším počtem kategorií,

- *„nevhodné pro velký počet kategorií, obsahující malý počet vzorků“* [2],
- pravděpodobně největší nevýhoda spočívá v nemožnosti vybudované stromy „přiučit“ něčemu novému (nové třídě) – lze pouze celý strom / les znovu vybudovat od začátku.

## 3.7 Náhodné lesy (Random Forests - RF)

Tato metoda je vlastně nadstavbou a konkrétní implementací rozhodovacích stromů z roku 2001 s použitím několika technik pro zvýšení použitelnosti na reálných datech. Kombinací těchto technik bylo dosaženo kvalitních výsledků a autoři (Leo Breiman a Adele Cutler) si pro tuto variantu zajistili ochranné známky Random Forests, RF, RandomForests, RandomForest a Random Forest. Název původně pochází z termínu *náhodné rozhodovací lesy*, který v roce 1995 nezávisle zavedl Tim Kam Ho.

### Základní princip

V základním principu se metoda neliší od rozhodovacích stromů, ale navíc zahrnuje vytvoření kolekce stromů (les) namísto jednoho stromu (tzv. „boosting“). Tento les rozhoduje hlasováním o přidělení datového bodu do třídy (v případě klasifikace) respektive výpočtem průměru cílové proměnné z odhadů jednotlivých stromů. Z tohoto důvodu je zde snaha o snížení generální chyby lesu jako celku namísto snahy o snížení nepřesnosti jednotlivých stromů. Dále je zde použit tzv. „bagging“ (nebo také „bootstrap aggregating“), který představuje náhodné rozdělení souboru trénovacích dat na $k$ částí. Jedna z částí se použije pro vlastní budování stromů, další se použijí pro ověření vybudovaného modelu (lesu).

Reálná data mohou obsahovat chybějící hodnoty, případně datové body s nekorektně přiřazenou třídou, a RandomForests si dle autorů dokáží s takovými situacemi poradit. [2][4][5]

### Autoři popisují následující výhody:

- výborné v přesnosti oproti ostatním současným algoritmům,
- efektivně zpracovává rozsáhlé datové kolekce,
- zvládá tisíce vstupních atributů bez jejich odstraňování,
- poskytuje odhady které atributy jsou při klasifikaci důležité,
- generuje vnitřní neovlivněné odhady celkové chyby lesa v průběhu jeho budování,
- má efektivní metodu pro odhadování chybějících dat a udržuje přesnost i v případě, že chybí velká část dat,
- poskytuje metodu pro vyrovnávání chyby v případě, že datová kolekce má nerovnoměrné rozdělení tříd,
- počítá prototypy, které poskytují informace o vztahu mezi atributy a klasifikací,

- počítá přiblížení mezi dvojicemi datových bodů, které může být využito pro shlukování,
- výše zmíněné vlastnosti mohou být rozšířený pro nepopsaná data, což vede k učení bez učitele,
- nabízí experimentální metodu pro detekci interakcí mezi atributy.

Nevýhody jsou v zásadě stejné jako u základní varianty – tedy rozhodovacích stromů 3.6 , nicméně musíme přihlédnout k omezení těchto nevýhod díky použitým vylepšením.

# 4 Srovnání metody rozhodovacích stromů s dalšími metodami strojového učení

Tato kapitola se vztahuje k metodám strojového učení popsaným v předchozí kapitole.

Jak již zde bylo zmíněno, žádná z metod strojového učení není dokonalá. Nelze tedy žádnou metodu aplikovat na všechny problémy a předpokládat, že ve všech případech bude výkonnost a kvalita zpracování nejideálnější. Proto je vhodné znát silné a slabé stránky jednotlivých metod v porovnání s dalšími metodami, abychom byli schopni vybrat nejvhodnější metodu pro konkrétní aplikaci.

Porovnání uvedené v Tabulka 2, vychází z bakalářské práce Barbory Pěchotové, jež byla zaměřena právě na srovnání těchto metod. Přesný odkaz na zdroj je uveden u popisku tabulky.

**Hodnocená kritéria metod**:

- Náročnost na použití – určuje jak náročné je použití metody na reálných datech.
- Předpoklady na rozložení – uvádí, zda má metoda předpoklad na rozložení.
- Tolerance k velkému počtu atributů – představuje schopnost metody zpracovávat data s velkým počtem atributů (neboli data o vysoké dimenzi).
- Typ atributů – jde o schopnost metody zpracovávat číselná či kategoriální (případně binární) data.
- Tolerance ke korelovaným proměnným – reprezentuje do jaké míry metoda toleruje navzájem nezávislé proměnné.
- Tolerance k odlehlým hodnotám – uvádí míru tolerance k odlehlým hodnotám, jež mohou mít značný vliv na výsledek.
- Schopnost klasifikovat do velkého počtu tříd.
- Tolerance k šumu – představuje míru tolerance ke vzorkům nepřinášejícím další informaci.
- Stabilita – stabilita metody je nepřímo úměrná míře ovlivnění výsledků malými změnami v původních datech.
- Přehlednost – neboli transparentnost výsledků metody
- Dostupnost v SW – dostupnost metody v současných statistických programech

Srovnání metody rozhodovacích stromů s dalšími metodami strojového učení

| metoda | **DA** | **LR** | **DT** | **k-NN** | **SVM** | **ANNs** | **RF** |
|---|---|---|---|---|---|---|---|
| **Náročnost na použití** | ** | * | ** | * | *** | *** | *** |
| **Předpoklady na rozložení** | ANO | ANO | NE | NE | NE | NE | NE |
| **Tolerance k velkému počtu atributů** | * | * | ** | *** | *** | *** | *** |
| **Typ atributů** | číselné + kategoriální | číselné + kategoriální | číselné + kategoriální | číselné | číselné | číselné + kategoriální | číselné + kategoriální |
| **Tolerance ke korelovaným proměnným** | * | * | *** | * | ** | ** | *** |
| **Tolerance k odlehlým hodnotám** | * | * | *** | ** | ** | *** | *** |
| **Schopnost klasifikovat do velkého počtu tříd** | ** | * | ** | * | ** | *** | *** |
| **Tolerance k šumu** | * | * | *** | * | *** | ** | *** |
| **Stabilita** | ** | ** | * | * | *** | ** | *** |
| **Přehlednost** | ** | ** | *** | ** | * | * | * |
| **Dostupnost v SW** | *** | *** | *** | *** | * | ** | * |

**Tabulka 2 - Srovnání klasifikačních metod** [2]

*** vysoká ** střední * nízká

# 5 Popis realizované implementace metody rozhodovacích stromů

Implementace, která je nedílnou součástí této práce, byla realizována v technologii .NET v jazyce C# jako konzolová aplikace. Implementace se nachází v podobě, která by byla snadno přetransformovatelná do podoby knihovny funkcí strojového učení metodou DT.

## 5.1 Úpravy základního principu metody pro efektivní použití na reálných datech

Jak již bylo zmíněno, základní princip byl zde teoreticky popsán pouze pro ideální data. Ta však v reálném světě nenajdeme, a proto je nutné, aby se prediktivní model dokázal s těmito „nestandardními“ situacemi vypořádat.

### Přílišná přesnost stromu

Přílišná přesnost, neboli přeučení modelu nebo také tzv. „overfitting“, se projevuje nízkou tolerancí vůči šumu. Řešením pro eliminaci tohoto chování je tzv. prořezávání („pruning“) stromu, které zároveň snižuje komplexitu generovaných stromů. V případě mé implementace jde prakticky o zavedení parametru „ImpurityThreshold“, který stanovuje práh entropie, při které se již podprostor dále nedělí. Stejný záměr je v zavedení parametru „MaxTreeDepth“, určující maximální počet úrovní budovaných stromů. Necílíme tedy na naučení stromu s entropií rovnající se 0 ve všech podprostorech. Nastavení hodnot těchto parametrů závisí na konkrétních datech Také technika, popsaná v následujícím odstavci, napomáhá omezit přeučení. [2][3]

### Vysoká variabilita a nízká stabilita

Vybudovaný rozhodovací strom velmi přesně „kopíruje“ poskytnutá trénovací data (což se může projevovat také přílišnou přesností popsanou v předchozím odstavci). Je ale malá pravděpodobnost, že bychom modelu při učení poskytli všechny přípustné datové body, aby klasifikoval se zanedbatelnou chybovostí, což je zároveň požadavek. Proto přichází ke slovu tzv. „bagging“. Toto slovo označuje metodu, při níž se místo jediného stromu buduje stromů několik - les. Každý ze stromů lesa je trénován na stejné kolekci trénovacích dat, ale pro jinou (náhodně vybranou) sadu atributů. Vstupní datový vektor $v$ prochází zpracováním každým ze stromů lesa. Výsledný výstup pro vektor $v$ je v případě klasifikace produktem hlasování stromů z celého lesa (třída s nejvyšším počtem hlasů) a v případě regrese se jedná o průměr výstupů jednotlivých stromů.

Obecně lze říci, že k zabránění přeučení nestačí pouze aplikace technik proti tomuto jevu, ale také je vhodné provádět učení na co možná nejrozsáhlejších (bohužel toto zvyšuje náročnost na výpočetní výkon a tím časovou náročnost procesu učení) a co nejméně zarušených trénovacích datech s vysokou měrou entropie (resp. rovnoměrnou četností jednotlivých tříd). Generalizace

(tedy „zobecňování“, čímž se s mírnou nadsázkou rozumí opak přeučení) je pak pro prediktivní model snazší. [2][3]

## 5.2 Pseudokód

Následující odstavce s pomocí pseudokódu popisující vnitřní fungování aktuální implementace.

### Načtení a předzpracování trénovacích dat

Po načtení CSV („comma separated values“ - čárkami oddělené hodnoty) souboru se v paměti počítače nachází pouze jejich textová kopie v jediném „kusu“ typu String (textový řetězec). Proměnné na řádcích 1 a 2 jsou globálními proměnnými, jsou tedy dostupné celému kódu. Na řádku 5 se nejprve provede rozdělení tohoto bloku na jednotlivé řádky (ve skutečné implementaci je dělicí výraz poněkud komplexnější, ale ní zde uveden, jelikož je zde popisován pouze princip) a pak v cyklu od řádku 7 jsou postupně hodnoty všech řádků rozděleny do polí. Cyklus od řádku 11 prochází jednotlivé atributy vektoru a u kategoriálních atributů si zaznamenává kolekci přípustných variant daného atributu.

```
1   datoveBody[]
2   seznamyVariant[][]
3
4   function LoadData(String data, char oddelovac)
5     radky[] = data.rozdelit('\n')
6
7     while ( !jePosledni(radky) )
8       i = 0
9       vektor[] = data.rozdelit(oddelovac)
10      datoveBody.pridat(vektor)
11      for ( atribut : vektor )
12        if ( jeKategorialni(atribut) )
13          if ( !jeVSeznamu(seznamyVariant[i], atribut))
14            seznamyVariant[i].pridat(atribut)
15            i = i + 1
16          endIf
17        endIf
18      endFor
19    endWhile
20  endFunction
```

## Vlastní učení

Následující pseudokód popisuje vnitřní implementaci pro učení klasifikačního stromu. Učení regresního stromu se oproti tomuto liší jen v několika detailech, které lze vyvodit z popisu metody rozhodovacích stromů v kapitole 3.6 .

Operace, provedená na řádku 2, vypočítá jaký počet atributů z jejich celkového počtu bude použito pro budování stromů. Náhodný výběr atributů na řádku 6 vybere náhodně podmnožinu atributů o velikosti, odpovídající hodnotě v proměnné `pocetAtributuProUceni`. Tento seznam atributů se následně v cyklu (od řádku 5) předává funkci pro vlastní budování stromu.

Funkce pro budování stromu si v prvním kroku (řádek 14) alokuje instanci nového stromu (tzn. třídy „TreeNode“) – v první iteraci reprezentuje kořen stromu. Dále dochází k alokaci proměnných, které budou uchovávat hodnoty té části algoritmu, která má za úkol optimální dělení prostoru datových bodů. 2 v sobě vnořené cykly pak procházejí všechny datové body, dostupné pro učení a všechny jejich atributy. Každá hodnota atributu (kolekce variant v případě kategoriálních dat) všech datových bodů představuje potenciální místo dělení prostoru. V každém z těchto bodů se spočítá entropie vzniklých podprostorů a z těch pak informační zisk $I_E$ (buduje-li se klasifikační strom) nebo redukce odchylky $I_V$ (pokud se buduje strom regresní). Podrobnější informace o výpočtu lze nalézt v kapitole 3.6 . Poté co jsou všechny body takto zpracovány, vyhodnotí se, ve kterém bodě byla hodnota $I_E$ resp. $I_V$ nejvyšší, podle čehož se vytvoří podmínka konkrétního uzlu. Pro tento dělicí bod se provede samotné rozdělení dat (tedy dat i hodnot cílových proměnných), jak je vidět na řádcích 35 a 36. Entropie již byly jednou spočteny a využijí se v dalším kroku, kterým je rozhodnutí (řádek 38) zda vytvořit list s konečným výsledkem pro daný podprostor, nebo generovat podstrom. Rozhodnutí závisí také na dalších hodnotách, které lze odvodit přímo z pseudokódu. Pokud je v tomto kroku rozhodnuto budovat podstrom, provede se to rekurzivním zavoláním této funkce s předáním pouze podmnožiny učicích dat a jejich tříd.

```
1    function Uceni()
2      pocetAtributuProUceni = zaokrouhlit(celkemAtributu * podíl
                                                          / 100)
3      les[]
4
5      for ( i : pozadovanyPocetStromu )
6        atributyProUceni[] =
     nahodneVyberAtributy(pocetAtributuProUceni, celkemAtributu)
7        les[i] = BudujStrom(atributyProUceni)
8      endFor
9
10     return les
11   endFunction
12
```

```
13  function BudujStrom(seznamAtributu,dataVProstoru,tridyVProstoru)
14    novyStrom = new uzel()
15    maximalniIe = -1; atributProDeleni; hodnotaAtributu
16    for ( atribut : seznamAtributu )
17      for ( datovyBod : dataVProstoru )
18        Ie = vypoctiInfoZisk(datovyBod, datovyBod[atribut])
19        if ( Ie > maximalniIe )
20          maximalniIe = Ie
21          atributProDeleni = atribut
22          hodnotaAtributu = datovyBod[atribut]
23        endIf
24      endFor
25    endFor
26
27    if (jeKategorialni(atributProDeleni))
28      for (varianta : seznamyVariant[atributProDeleni]=>index)
23        novyStrom.podminka[index] = „vstupniVektor==varianta“
30      endFor
31    else
32      novyStrom.podminka = „ vstupniVektor > hodnotaAtributu“
33    endIf
34
35    podprostoryDat[][] = rozdelitData(dataVProstoru, datovyBod,
                                        datovyBod[atribut])
36    podprostoryTrid[][] = rozdelitTridy(tridyVProstoru,
                              datovyBod, datovyBod[atribut])

37    for ( podprostor : podprostoryDat => index )
38      if ( vypoctiEntropii(podprostor) > minimalniEntropie AND
            urovenStrmou < maximalniUrovenStromu AND )
39        novyStrom.vetev[index] = BudujStrom(seznamAtributu,
                  podprostoryDat[index], podprostoryTrid[index])
40      else
41        novyStrom.list =
                    nejpocetnejsiTrida(podprostoryTrid[index])
42      endIf
43    endFor
44
45    return novyStrom
46  endFunction
```

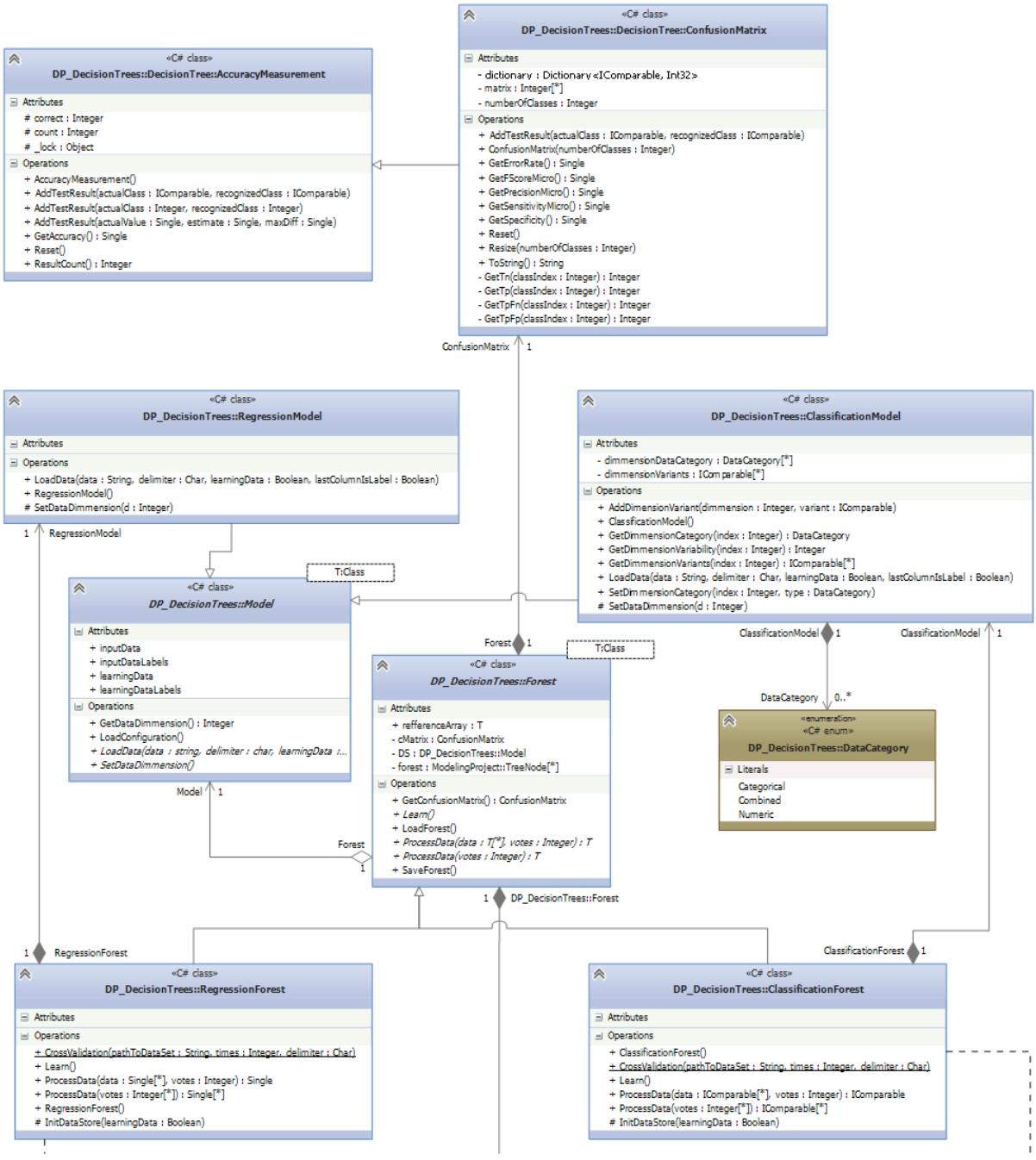
### Zpracování dat vybudovaným lesem

Následující pseudokód popisuje vnitřní implementaci pro zpracování dat klasifikačním stromem. Zpracování dat regresním stromem se oproti tomuto liší jen v několika detailech, které lze vyvodit z popisu metody rozhodovacích stromů v kapitole 3.6 .

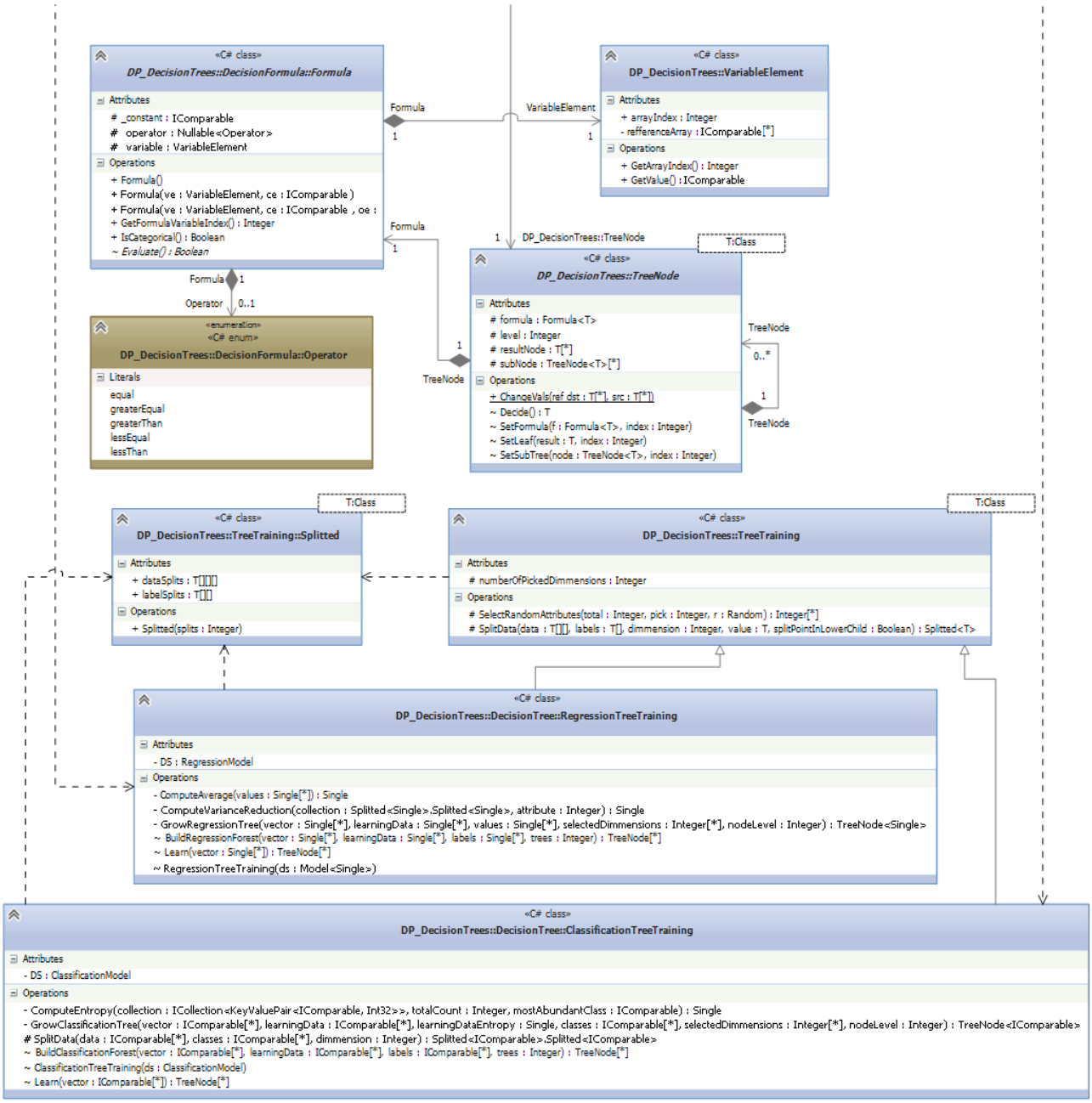
Pole `stromy` je globálním polem a je proto dostupné v celé aplikaci. Cyklus na řádku 6 iteruje nad všemi vektory z dvojrozměrného vstupního parametru `vstupniDatoveBody`. Funkce vrací pole „výsledky“ s třídami, do kterých les zařadil jednotlivé vstupní datové body. Analogicky pro regresi bude vrácené pole `vysledky` obsahovat odhady hodnot.

```
1   stromy[]
2
3   function ZpracovatData(vstupniDatoveBody[][])
4     vysledky[]
5
6     for ( vektor : vstupniDatoveBody => indexVektoru )
7       posliVektorDoReferencnihoVektoru(vektor)
8
9       dilciVysledky[]
10      for ( strom : stromy => indexStromu )
11        dilciVysledky[indexStromu] = strom.klasifikuj()
12      endFor
13
14      vysledky[indexVektoru] =
                          najdiNejcetnejsiVysledek(dilciVysledky)
15    endFor
16
17    return vysledky
18  endFunction
```

## 5.3 Třídní diagram




**Obrázek 9 - Třídní diagram (horní polovina)**



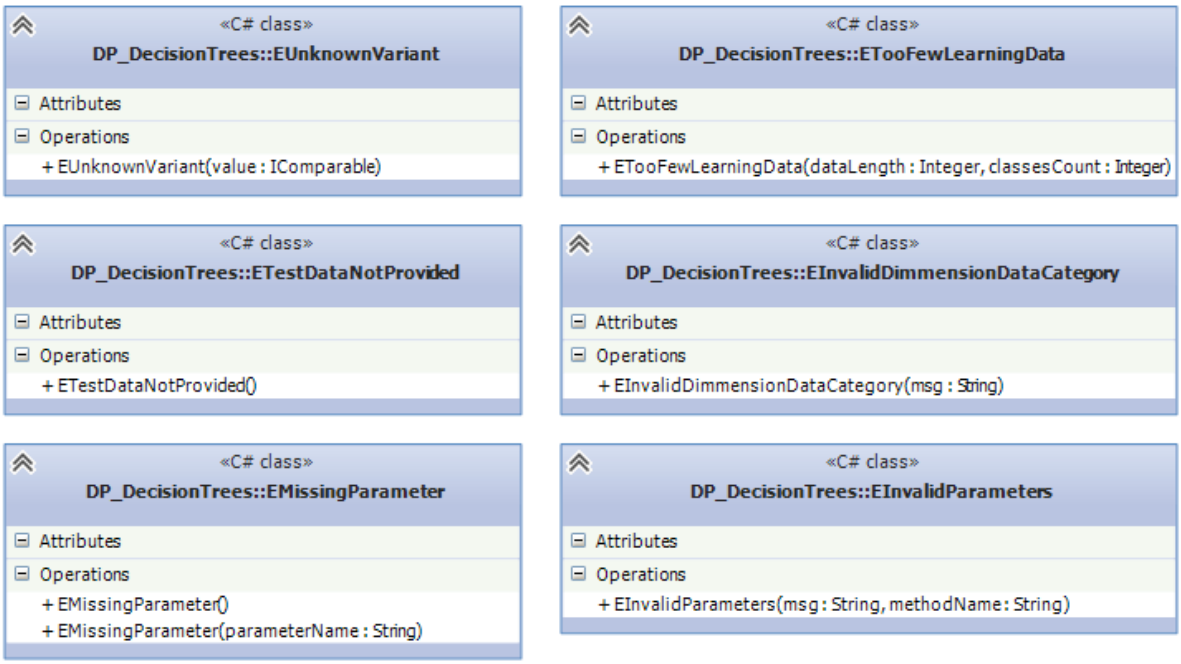

**Obrázek 10 - Třídní diagram (dolní polovina)**

**Popis jednotlivých tříd:**

- **Program** – vstupní třída programu,

- **Model<T>** - generická třída zpracovávající a uchovávající trénovací data (včetně výsledků – tříd pro klasifikaci či spojitých hodnot pro regresi), vstupní data (i s případnými výsledky) a také konfiguraci pro trénovací algoritmus,
- **RegressionModel** – představuje rozšíření třídy „Model<T>“ pro data spojitého charakteru, kdy je úkolem prediktivního modelu regrese,
- **ClassificationonModel** – představuje rozšíření třídy „Model<T>“ pro data spojitého nebo kategorického charakteru, kdy je úkolem prediktivního modelu klasifikace,
- **VariableElement** - prvek logické formule představující referenci na aktuální hodnotu konkrétního atributu právě zpracovávaných dat,
- **Formula** – objekt představující logickou formuli a skládající se z proměnného elementu (instance třídy VariableElement - viz. bod výše), konstanty, vůči které je porovnávána hodnota proměnného elementu, a operátoru (operátor může nabývat některé z výčtu hodnot enumerátoru „Operator“ - menší než, menší nebo roven, větší než, větší nebo roven, rovnající se, anebo může být prázdný – v případě že se jedná o porovnávání s kategorickým atributem),
- **TreeNode<T>** – generický objekt rozhodovacího stromu obsahující $N$ logických formulí (viz. další bod) a až $N+1$ podstromů a listů (kombinovaně) pro různé výsledky vyhodnocení formule. U spojitých dat je $N=1$ podprostor dat dělíme vždy na právě 2 podprostory, $N>1$ v případě, že se jedná o vyhodnocování kategorických dat, kde se $N$ rovná počtu možných variant kategorického atributu, na který se odkazuje proměnný prvek (tj. instance třídy „VariableElement“).
- **TreeTraining<T>** - tato generická třída se stará o vše kolem vlastního učení, resp. budování stromů, potažmo lesa,
- **RegressionTreeTraining** – představuje rozšíření třídy „TreeTraining<T>“ pro data spojitého charakteru, kdy je úkolem prediktivního modelu regrese. Největší odlišnost této třídy od třídy ClassificationTreeTraining spočívá v odlišném způsobu výpočtu nejideálnějšího místa pro dělení prostoru. Více o tomto lze nalézt v kapitole 3.6 .
- **ClassificationonTreeTraining** – představuje rozšíření třídy „TreeTraining<T>“ pro data spojitého nebo kategorického charakteru, kdy je úkolem prediktivního modelu klasifikace. Největší odlišnost mezi toto třídou a třídou RegressionTreeTraining spočívá v odlišném způsobu výpočtu nejideálnějšího místa pro dělení prostoru. Více o tomto lze nalézt v kapitole 3.6 .
- **AccuracyMeasurement** – jednoduchá třída pro záznam počtu správných výsledků (správné zařazení do tříd, či regresní odhad hodnoty v toleranci) pro výpočet úspěšnosti prediktivního modelu,

- **ConfusionMatrix** – rozšíření třídy AccuracyMeasurement, jež uchovává výsledky rozhodnutí vygenerovaných stromů pro jednotlivá data s porovnáním, zda strom zařadil data do správné třídy. Z těchto údajů lze vypočítat různé míry vypovídající o přesnosti a kvalitě prediktivního modelu – více lze nalézt v kapitole 6.2 .




**Obrázek 11 - Třídní diagram (třídy výjimek)**

**Význam výjimek v implementaci:**

- **EUnknownVariant** – tato výjimka je vyvolána v případě, že některý kategorický atribut má ve vstupních datech variantu, která nebyla v daném atributu přítomna v trénovacích datech, a proto se s ní nepočítalo,
- **ETooFewLearningData** – vyvolání této výjimky indikuje nízký počet trénovacích datových bodů, jejich počet je menší než je počet tříd pro klasifikaci,
- **ETestDataNotProvided** – vyvolání této výjimky nastává v případě, kdy je požadováno zpracování dat, ale vlastní data pro zpracování nebyla poskytnuta,
- **EInvalidDimmensionDataCategory** – typ atributu (kategorický/spojitý) byl u některého z atributů zvolen chybně,
- **EMissingParameter** – vyvolání této výjimky nastane v situaci, když v konfiguračním souboru chybí definice některého z parametrů aplikace (viz. Kapitola 5.4 ),

- **EInvalidParameters** – tato výjimka je vyvolána, pokud je požadován výběr většího počtu atributů, než je celkový počet atributů trénovacích dat.

## 5.4 Parametry nastavení aplikace

Parametry nastavení aplikace, umístěné v souboru App.config, mají vliv na algoritmus učení. Určují několik důležitých hodnot, které ovlivňují výslednou kvalitu vybudovaného prediktivního modelu – lesa.

- **TreesInForest** – určuje počet stromů v lese.
- **MaxTreeDepth** – tento parametr omezuje maximální úroveň uzlu stromu, což pomáhá předcházet přeučení modelu; celé číslo v rozsahu 5÷30.
- **AtributePercentage** – parametr určuje procentuálně vůči datům počet atributů, který bude použit při učení jednotlivých stromů; desetinné číslo v intervalu <0,1;1>.
- **ImpurityThreshold** – tento parametr má význam pouze při učení klasifikačního lesa a určuje práh entropie, pod kterým již nedochází k dalšímu dělení prostoru, čímž pomáhá předcházet přeučení modelu; desetinné číslo v intervalu <0,1;1>.
- **MinTreeSuccessRate** – jelikož jsou atributy, nad kterými proběhne učení stromu, vybírány náhodně, může se stát, že vygenerovaný model nebude dostatečně přesně „kopírovat“ trénovací data. Pokud jeho úspěšnost při ověření na trénovacích datech klesne pod tuto úroveň, tento jedinec (strom) bude zrušen a nahrazen nově vygenerovaným stromem; desetinné číslo v intervalu <0,51;1>.
- **AttributesCount** – udává počet očekávaných atributů dat.
- **ItemsCount** – udává počet očekávaných tříd, do kterých mají být data tříděna (jedná-li se o klasifikaci), resp. udává maximální počet bodů cílové proměnné, ze které má být vypočten průměr pro odhad (v případě regrese).
- **DataCategory** – nabývá jedné z hodnot Numeric (pro spojitá data) / Categorical (pro data kategorická, speciálním případem takovýchto dat jsou data binární) / Combined (kombinace spojitých a kategorických).

Příklad obsahu konfiguračního souboru App.config lze vidět zde:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>

  <!-- configuration for handwritten digits data set -->
  <appSettings>
    <add key="TreesInForest" value="40" />  <!-- size of forest -->
     <add key="MaxTreeDepth" value="14" />  <!-- maximum depth level of generated
trees (5 ~ 30) -->
      <add key="AttributePercentage" value="45" />  <!-- how many dimensions of
input to use in a single tree (in percentage of input vector dimension count) -->
    <add key="ImpurityThreshold" value="0.18" />  <!-- maximal entropy (impurity)
of child split to assign class (0.001 ~ 0.49) -->
     <add key="MinTreeSuccessRate" value="0.82" />  <!-- minimal success rate to
not eliminate the tree from forest (0.51 ~ 1) -->
    <add key="AttributesCount" value="256"/>  <!-- dimmension of input values -->
      <add key="ItemsCount" value="10" />    <!-- number of classes into which
should be input split (classification) / maximal amount of points for leaf mean
(regression) -->
      <add key="DataCategory" value="Categorical" />      <!-- allowed options:
Numeric / Categorical / Combined -->
  </appSettings>
</configuration>
```

# 6 Měření výkonu realizované implementace

Abychom dostali objektivní přehled o výkonnosti algoritmu a o kvalitě modelu, je nutné provést měření, kdy jsou aplikaci předána reálná data a měří se několik údajů, vypovídajících o určitých vlastnostech řešení. Tato kapitola obsahuje zjištěné výsledky týkající se prostorové a časové složitosti a měření kvality klasifikace dvěma odlišnými způsoby.

Všechny experimenty a měření byly provedeny na reálných kolekcích dat pocházejících z knihovny UCI [24], které byly upraveny jen minimálně – změna původního formátu na formát CSV.

Veškeré testy probíhaly na počítačovém systému s touto konfigurací:

- přenosný počítač HP ProBook 4520s,
- procesor Intel i3 M370 – 2,4GHz (2 fyzická jádra, běh až 4 vláken simultánně),
- 8GB RAM DDR3 1066MHz,
- pevný disk s 5400ot./min,
- operační systém Windows 7 64-bit.

Datové sady původem z knihovny UCI:

| Sada | Název sady | Počet atributů | Počet datových bodů | Typ dat |
|---|---|---|---|---|
| A | Semeion Handwritten Digit | 256 | 1593 | Celočíselný |
| B | Wine Quality | 12 | 4898 | Spojitý |
| C | Auto MPG | 8 | 398 | Kategorický, spojitý |

**Tabulka 3 - Datové sady pro měření**

## 6.1 Časová a prostorová náročnost

Objektivní měření časové náročnosti není právě triviální úkol. Jako nejvhodnější se jeví varianta měření procesorového času, což je součet časů, po jaké jednotlivá vlákna využívala procesor počítače.

Prostorová náročnost je v této práci uváděna jako přibližná maximální velikost využívané operační paměti, zjištěná systémovým nástrojem OS Windows „Správce úloh“.

Naměřené hodnoty obou veličin jsou v rámci této práce uváděny přímo u jednotlivých typů testů a experimentů.

## 6.2 Vliv jednotlivých parametrů, matice chybovosti

Pro zkoumání vlivů nastavení parametrů na výkonnost a časovou a prostorovou náročnost klasifikace byla využita mimo jiné tzv. „confusion matrix“ nebo také „error matrix“ (česky možno „matice chybovosti“). Jedná se o speciální typ kontingenční tabulky, ze které je možné získat pomocí výpočtů různé hodnotnější statistické údaje než je obyčejné měření úspěšnosti, které by nemělo dostatečně vypovídající hodnotu, pokud by byly třídy v datech zastoupeny výrazně nerovnoměrně. Mezi kvalitní ukazatele patří tzv. sensitivita (udává stupeň korektních označení danou třídou) a tzv. specificita (ta naopak představuje korektní neoznačení danou třídou), jejichž výpočet je popsán níže v této kapitole.

Matice chybovosti je čtvercová o velikosti M×M, kde M = počet tříd, do kterých se klasifikuje. Řádky matice představují počty skutečných tříd, kam jednotlivá data patří a sloupce pak počty tříd, do kterých byly datové body zařazeny. Příklad je znázorněn v Tabulka 4. Je zde patrné např. že třikrát byl pes označen za kočku, sedmkrát byl označen správně a dvakrát byl označen jako králík.

| | | Zařazení do tříd modelem | | |
|---|---|---|---|---|
| | | Kočka | Pes | Králík |
| Skutečné třídy | Kočka | 8 | 1 | 0 |
| | Pes | 3 | 7 | 2 |
| | Králík | 1 | 4 | 12 |

**Tabulka 4 - Příklad matice chybovosti** [20]

Pro každou třídu lze vypočítat několik dílčích údajů:

- TP („true positive“) - korektní určení, že se jedná o danou třídu.
- TN („true negative“) - korektní určení, že se nejedná o danou třídu.
- FP („false positive“) - nekorektní určení, že se jedná o danou třídu.
- FN („false negative“) - nekorektní určení, že se nejedná o danou třídu.

Z těchto údajů je již možné vypočítat senstitivitu (TPR - „true positive rate“)

$$TPR = \frac{\sum_{i}^{l} tp_i}{\sum_{i}^{l} tp_i + fn_i}$$

a specificitu (TNR - „true negative rate“)

$$TNR = \frac{\sum_{i}^{l} tn_i}{\sum_{i}^{l} tn_i + fp_i}$$

$l$ – počet tříd, do který se data třídí (klasifikují)

Tyto údaje budou využity při následujících měřeních. [19][20]

### Testování reálných dat – klasifikace

Sloupec „parametry“ v následujících tabulkách je zkráceným zápisem pro parametry TreesInForest/MaxTreeDepth/AttributePercentage/ImpurityThreshold.

První test proběhl na sadě kategorických dat A a výsledky pro 6 běhů s různým nastavením parametrů učení lze vidět v Tabulka 5. Z celé sady dat bylo 1200 datových bodů použito pro učení. U žádného z běhů nebylo využití operační paměti vyšší než 20800kB.

| | procesorová doba učení [ms] | procesorová doba zpracování [ms] | specificita | sensitivita | parametry |
|---|---|---|---|---|---|
| Běh 1 | 20015,0 | 62,4 | 0,987 | 0,893 | 40 / 14 / 45 / 0,18 |
| Běh 2 | 24850,0 | 78,0 | 0,986 | 0,885 | 40 / 16 / 55 / 0,18 |
| Běh 3 | 43446,0 | 77,0 | 0,987 | 0,891 | 70 / 16 / 55 / 0,18 |
| Běh 4 | 28501,0 | 73,6 | 0,988 | 0,896 | 50 / 15 / 15 / 0,10 |
| Běh 5 | 19032,0 | 47,0 | 0,984 | 0,875 | 30 / 14 / 50 / 0,09 |
| Běh 6 | 36364,0 | 31,0 | 0,959 | 0,720 | 35 / 17 / 95 / 0,05 |

**Tabulka 5 - Statistika testování klasifikace kategorických dat**

Druhý test běžel na sadě spojitých dat B. Výsledky šesti běhů tohoto testu s různým nastavením parametrů učení lze vidět v Tabulka 6. Z celé sady dat bylo 3500 datových bodů použito pro učení. U žádného z běhů nebylo využití operační paměti vyšší než 15200kB.

Měření výkonu realizované implementace

| | procesorová doba učení [min:s] | procesorová doba zpracování [ms] | specificita | sensitivita | parametry |
|---|---|---|---|---|---|
| Běh 1 | 22:21 | 93,6 | 0,810 | 0,516 | 20 / 14 / 65 / 0,14 |
| Běh 2 | 33:42 | 140,4 | 0,826 | 0,543 | 40 / 16 / 55 / 0,18 |
| Běh 3 | 19:49 | 187,2 | 0,822 | 0,537 | 70 / 16 / 55 / 0,18 |
| Běh 4 | 24:44 | 171,6 | 0,811 | 0,517 | 50 / 15 / 15 / 0,10 |
| Běh 5 | 5:32 | 15,6 | 0,732 | 0,406 | 30 / 14 / 50 / 0,09 |
| Běh 6 | 38:39 | 234,0 | 0,881 | 0,553 | 35 / 17 / 95 / 0,05 |

**Tabulka 6 - Statistika testování klasifikace spojitých dat**

### Shrnutí vlivu nastavení hodnot parametrů

Při měřeních bylo možné pozorovat, jaký vliv na kterou vlastnost mají jednotlivé parametry aplikace. Optimální nastavení parametrů může poměrně zásadně ovlivnit celkovou výkonnost algoritmu i vygenerovaného prediktivního modelu.

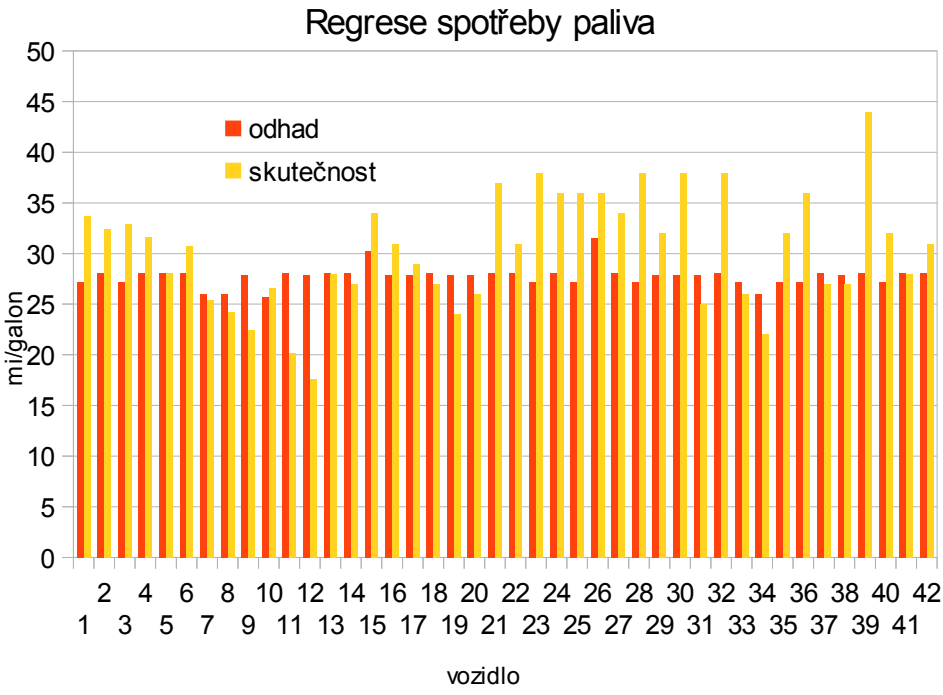
### Testování reálných dat – regrese

Testování na datové sadě C (spojitá data, týkající se spotřeby paliva osobních aut) s těmito parametry:

- TreesInForest (počet stromů v lese) = 15
- MaxTreeDepth (maximální úroveň stromu) = 18
- AttributePercentage (procentuální množství atributů, používané při učení) = 30
- MinTreeSuccessRate (akceptační kritérium minimální úspěšnosti na trénovacích datech pro přijetí stromu) = 0,766
- z celé sady bylo 350 datových bodů použito pro učení
- další atributy prakticky popisují data jako taková a nelze jimi ovlivnit výkonnost, proto zde nejsou uvedeny

V grafu na Obrázek 12 lze vidět jak vygenerovaný model odhadoval spotřebu paliva osobních automobilů a jaká byla skutečná spotřeba.

Učení modelu trvalo 7min:29,7s a zpracování dat pak pouhých 31ms. Maximální využití operační paměti nepřesáhlo 6800kB.




**Obrázek 12 - Graf výstupu a skutečných hodnot při testu regrese**

## 6.3 Měření úspěšnosti klasifikace pomocí N-násobné krosvalidace

N-násobná krosvalidace je technika vyhodnocování kvality klasifikace, při které je trénovací sada dat náhodně zamíchána, a poté rozdělena na N podmnožin o přibližně stejném počtu datových bodů. Poté se postupuje tak, že se prvních N-1 podmnožin použije pro trénování klasifikátoru a poslední podmnožina je použita jako testová, přičemž se měří sensitivita a specificita. V následující iteraci je postup opakován, ale pro trénování se použije N-1 podmnožin přičemž se vynechá jiná podmnožina, která bude použita pro testování. Takto proběhne N iterací a pokaždé bude jiná podmnožina použita pro testování. Existují také další speciální typy této metody, o kterých se lze dočíst v uvedených zdrojích. [21][22]

Testová sada A byla podrobena testu 10-násobné krosvalidace s těmito výsledky (představují průměry měřených hodnot):

- sensitivita = 0,907
- specificita = 0,989
- celkový procesorový čas ≈ 4min:15s
- prostorová náročnost ≈ 57500kB

# 7 Zhodnocení výsledků a závěr

Cílem této diplomové práce bylo seznámení se s metodami pro analýzu dat - klasifikaci a regresi, zejména se zaměřením na metodu rozhodovacích stromů. Bylo navrženo a implementováno takové řešení rozhodovacích stromů, které dokáže předzpracovaná data ve formátu CSV (posledním sloupcem odpovídá třídě, do které datový bod patří / hodnotě cílové proměnné pro regresi) analyzovat a následně vygenerovat prediktivní modely, které zpracovávají data binární, kategorická, celočíselná i spojitá, případně i jejich kombinace. Předzpracovanými daty se rozumí např. (u obrázků s ručně psanými číslicemi) na maximum zvýšený kontrast, následné zjednodušení v podobě nahrazení černých bodů jedničkou, a bílých nulou. Nakonec byl obrázek, původně reprezentovaný maticí bodů, „narovnán“ do řady hodnot, aby jej bylo možno uložit do CSV formátu. Implementace je připravena k použití jako knihovna s možností volby parametrů přes konfigurační soubor.

## 7.1 Vymezení vhodných úloh a stanovení limitů použití aplikace

Implementace obsahuje kromě vlastní realizace rozhodovacích stromů také metody pro měření kvality predikčního modelu a časové náročnosti učení a zpracování dat. Toto bylo využito při experimentech a měřeních, jejichž výsledky lze nalézt výše v kapitole 6 .

Z realizovaných experimentů lze usoudit, že pro danou implementaci je nejvhodnějším úkolem klasifikace kategorických dat. Při úloze klasifikace dosahuje algoritmus velmi dobrých výsledků (nejvyšší úspěšnost a také řádově vyšší rychlost učení). Při analýze a zpracování dat spojitých jsou celkové výsledky dle očekávání horší. Časová náročnost tohoto úkolu je z jejího principu vyšší, a tak trvá učení podstatně déle. Nižší úspěšnost modelu u tohoto typu úkolu může být do značné míry způsobena obtížným nastavením hodnot parametrů aplikace. Samotná klasifikace i regrese vygenerovanými stromy je dle očekávání velmi rychlá se všemi typy zpracovávaných dat.

## 7.2 Možnosti vylepšení realizované implementace

Tato kapitola obsahuje návrhy na zlepšení konkrétní implementace a to hned z několika hledisek, které by rozšiřovaly možnosti použití a především kvalitu predikce a výkonnost. Některé z návrhů vycházejí z přítomnosti těchto vylepšení v existujících řešeních, jiné představují vlastní nápady, podpořené aplikací znalostí získaných z výuky na VŠB.

### Paralelizace algoritmu

Paralelizace algoritmu přináší významnou možnost zvýšení rychlosti učení i vlastního zpracovávání dat a to při možnostech současného (rok 2015) hardwaru.

Pro rodinu algoritmů rozhodovacích lesů jsou možné dva přístupy – buď paralelizovat algoritmus na úrovni lesa, takže by budování *t* stromů, resp. zpracování dat pomocí *t* stromů, probíhalo v *t* vláknech. Tento přístup využívá toho, že jednotlivé stromy jsou na sobě nezávislé jak při budování, tak při zpracování dat. Implementovat tento přístup na CPU je relativně jednoduché. Nevýhodou tohoto přístupu pak je vyšší prostorová náročnost než u následujícího přístupu, dále nevhodnost pro paralelizaci metod DT, kde jsou využívány pouze jednotlivé stromy a také v klesajícím nárůstu rychlosti, pokud počet stromů v lese bude menší než počet dostupných výpočetních jednotek (resp. maximálního možného počtu simultánně běžících vláken na daném počítačovém systému).

Paralelizace na úrovni jednotlivých stromů má smysl především, pokud v implementaci budujeme jediný strom, protože předchozí varianta by se v takové situaci zcela minula účinkem. U tohoto přístupu dojde k paralelizaci při počítání informačního zisku nad každým z bodů v děleném prostoru. Nevýhoda spočívá pravděpodobně jen ve složitější implementaci než v případě předchozího přístupu.

Například Benedikt Waldvogel se ve své diplomové práci zabýval akcelerací metody Náhodných Lesů pomocí GPU. Jeho práce je dostupná zde: http://www.ais.uni-bonn.de/theses/Benedikt_Waldvogel_Master_Thesis_07_2013.pdf.

### Chybějící data

Ne vždy máme k dispozici zcela kompletní data (například u dotazníkového šetření nebyl některý z údajů respondentem vyplněn, data na médiu by poškozena apod.), a proto by bylo užitečné, kdybychom se dokázali s takovými chybějícími údaji vypořádat. Rozdílně se doplňují data chybějící v tréninkové sadě oproti těm v testové sadě.

### Chybějící data v tréninkově sadě

Není-li atribut s chybějícími daty *A* kategorický, lze chybějící data třídy *T* atributu *A* doplnit mediánem ze všech hodnot atributu *A*, které patří do třídy *T*. Analogicky pro kategorická data lze dosazovat za chybějící hodnoty nejčastěji se vyskytující neprázdnou hodnotu, která patří atributu *A* a spadá do třídy *T*.

Druhý způsob je výpočetně náročnější, avšak s lepšími výsledky a to i pro velké množství chybějících dat. Je-li x(m,n) chybějící spojitá hodnota, odhadneme její vyplnění jako průměr z nechybějících hodnot m-tých atributů, vážené proximitou mezi n-tým vektorem a vektorem s nechybějící hodnotou. V případě chybějící kategoriální hodnoty nahradíme tuto hodnotou s nejvyšší frekvencí, kde frekvence je vážena proximitou. Tímto způsobem nejprve doplníme hrubé a nepřesné výsledky, následně se provede běh lesa a spočítají se proximity. [4]

Proximitu (neboli „blízkost“ či obrácenou hodnotou vzdálenosti dvou bodů v prostoru) vypočteme pomocí vzorce:

$$prox(A, B) = \frac{1}{|AB|} = \frac{1}{\sqrt{\sum_{i=1}^{D} (b_i - a_i)^2}}$$

$D$ – dimenze prostoru (počet atributů)

$a_i$, $b_i$ – i-tá souřadnice bodu A, resp. bodu B

## Chybějící data v testové sadě

Chybí-li data v testové sadě dat, pak máme dvě možnosti v závislosti na tom, zda je k danému datovému bodu s chybějící hodnotou k dispozici také třída, do které patří.

Pokud máme k dispozici třídy k datům, nahradíme je hodnotami použitými při nahrazování v tréninkové sadě dat.

V opačném případě je každý datový bod zkopírován tolikrát, kolik je různých tříd. První kopii je přiřazena třída 1, druhé kopii třída 2 atd. Tento rozšířený obor testových dat projde zpracováním vybudovaným lesem. V každé ze sad kopií je právě ta s nejvíce hlasy stanovena jako třída původních dat. [4]

## Oprava nekorektního přiřazení datových bodů do tříd v tréninkové sadě dat

U některých tréninkových sad bývá roztřídění dat do tříd provedeno lidským rozhodováním, což může v některých případech vést ke značnému výskytu nekorektně klasifikovaných dat. Mnoho takových chyb může být nalezeno pomocí tzv. odchylkové míry („outlier measure“).

Odchylky můžeme obecně definovat jako případy datových bodů, jejichž blízkosti od ostatních jsou malé (jinými slovy: vzdálenosti jsou velké). Užitečné je také nalézt takové výchylky pro jednotlivé třídy zvlášť. Definujme průměrnou blízkost datového bodu $n$, který patří do třídy $t$, od ostatních datových bodů ve třídě $t$:

$$\bar{P}(n) = \sum_{cl(u)=t} prox^2(n, u)$$

Hrubá odchylková míra pro datový bod $n$ je pak definována takto: $o_r = count(t) / \bar{P}(n)$
Toto může být velmi velké číslo, pokud je průměrná blízkost malá (resp. průměrná vzdálenost velká). Proto potřebujeme nalézt také medián těchto hodnot a jejich absolutní odchylku od mediánu. Abychom získali finální odchylkovou míru, odečteme medián od každé hodnoty $o_r$ a dělíme hodnotou absolutní odchylky. [4]

## Učení bez učitele (shlukování)

Při učení bez učitele sestávají data ze sady vektorů stejné dimenze bez označení tříd. Není zde žádné vodítko, proto je pole zcela otevřené pro nejednoznačné závěry. Obvyklým cílem je shlukování dat. Uvidíme, zda jednotlivé blízké datové body spadají do stejných shluků, kterým by bylo možné přidělit určitý význam.

Postup, například u Náhodných Lesů (RF), spočívá v označení originálních dat jako třída 1 a vytvoření umělé třídy stejné velikosti, označené jako třída 2. Uměle vytvořená druhá třída vzniká náhodným vzorkováním jednorozměrného rozdělení originálních dat. Každý prvek třídy 2 je pak vytvořen ze souřadnic, kde první souřadnice je ze vzorkovaných *N* hodnot {x(1,n)}, druhá souřadnice je nezávisle vzorkována z *N* hodnot {x(2,n)} a tak dále. Třída 2 má proto rozdělení nezávislých náhodných atributů, každý mající stejné jednorozměrné rozdělení, tak jako odpovídající atribut v originálních datech. Třída 2 tak ruší strukturu závislostí s originálními daty. Nyní zde máme problém o dvou třídách, který již můžeme zpracovat rozhodovacími stromy (lesy), což umožňuje použít rozhodovací stromy (lesy) na původní, neoznačený datový soubor. [4]

## Optimalizace hledání ideálního nastavení parametrů pomocí biologicky inspirovaných výpočtů

Biologicky inspirované výpočty jsou kapitolou samy o sobě. Jejich inspirace v přírodě dokáže velké množství výpočetně náročných problémů (často optimalizačních) urychlit skokově o několik řádů díky zcela odlišnému přístupu k řešení dané úlohy.

Do jisté (velmi malé) míry současné řešení již využívá jistou heuristickou techniku a to když algoritmus pro učení stromu náhodně vybírá podmnožiny atributů trénovacích dat, které využije při učení stromů lesa.

V kapitole 6.2 je zmíněno, že nastavení parametrů aplikace má vliv jak na kvalitu vybudovaného prediktivního modelu (úspěšnost predikce), tak na rychlost učení i zpracování dat a v neposlední řadě také na prostorovou náročnost. Proto se zde otevírají poměrně široké možnosti optimalizace.

Potenciálně vhodnou bioinspirovanou metodou by mohla být diferenciální evoluce. Při jejím použití bychom považovali za jedince jednotlivé instance tříd, obstarávající učení (třída „TreeLearning“) a za jednotlivé parametry jedinců bychom dosadili parametry realizované aplikace. Zbývá poslední substituce – tzv. fitness hodnota, neboli vyjádření kvality jedince. Tuto bychom získali, kdybychom s daným nastavením parametrů spustili proces učení na vzorku cílového typu dat a podle preferencí bychom fitness hodnotu vypočítali jako kombinaci úspěšnosti vygenerovaného prediktivního modelu, časové složitosti a případně dalších metrik. Zvláštností pak by bylo použití diferenciální (nebo jiné) metody „ve druhé úrovni“ k nalezení nejoptimálnějšího nastavení parametrů diferenciální evoluce.

Potenciálně vhodných bioinspirovaných metod je více, např. slepý algoritmus (který ale představuje naprosto základní a velmi neefektivní metodu), horolezecký algoritmus, tabu search, genetické algoritmy či SOMA (SamoOragnizující se Migrační Algoritmus). [23]

# 8 Seznam příloh umístěných na DVD

1) Zdrojové kódy implementace
2) Testová data použitá při vývoji a na kterých byla prováděna také měření a experimenty

## Literatura a zdroje dat


[1] ABU-MOSTAFA, Yaser S., Malik MAGDON-ISMAIL a Hsuan-Tien LIN. 2012. *Learning from data: a short course*. s.l.: AMLBook, xii, 201 s. ISBN 978-1600490064.

[2] PĚCHOTOVÁ, Barbora. *Srovnání klasifikačních metod pro aplikaci na biologických datech* [online]. Brno, 2009. Dostupné z: http://is.muni.cz/th/243713/prif_b/BP_Pechotova.doc. Bakalářská práce. Masarykova Univerzita. Vedoucí práce Mgr. Klára Kubošová.

[3] Decision tree learning. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- . Dostupné z: http://en.wikipedia.org/wiki/Decision_tree_learning.

[4] BREIMAN, Leo a Adele CUTLER. Random Forests. *University of California, Berkeley: Department of Statistics* [online]. 2001. Dostupné z: http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm.

[5] Random forest. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- . Dostupné z: http://en.wikipedia.org/wiki/Random_forest

[6] WILKINSON, Leland. *Classification and Regression Trees* [online]. Illinois, USA, 2012. Dostupné z: http://cda.psych.uiuc.edu/multivariate_fall_2012/systat_cart_manual.pdf, University of Illinois.

[7] JIŘINA, Marcel. *Jak na neuronové sítě v programu STATISTICA - neuronové sítě*. 1. vyd. Praha: StatSoft, 2003, 73 s. ISBN 978-80-904033-0-7.

[8] Artificial neural network. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- . Dostupné z: http://en.wikipedia.org/wiki/Artificial_neural_network.

[9] K-nearest neighbors algorithm. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- . Dostupné z: http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[10] Diskriminační analýza. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: http://cs.wikipedia.org/wiki/Diskriminační_analýza.

[11] HOSMER, David W. 2000. Applied logistic regression. 2nd ed. New York: John Wiley, 375 s. ISBN 0-471-35632-8.

[12] Logistic regression. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: http://en.wikipedia.org/wiki/Logistic_regression.

[13] Support vector machine. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: http://en.wikipedia.org/wiki/Support_vector_machine.

[14] SHAPIRO, Linda a STOCKMAN. *Pattern Recognition Concepts*. Computer Vision [online]. Prentice-Hall, 2001. Dostupné z: https://courses.cs.washington.edu/courses/cse576/book/ch4.pdf.

[15] SHAPIRO, Linda. *Information Gain*. 2001, 13 s. Dostupné z: https://courses.cs.washington.edu/courses/cse455/10au/notes/InfoGain.pdf.

[16] DE FREITAS, Nando. CPSC540: Decision trees. In: [online]. 2013. Dostupné z: http://www.cs.ubc.ca/~nando/540-2013/lectures/l8.pdf.

[17] What are decision trees?. *National Center for Biotechnology Information* [online]. 2008 [cit. 2015-03-22]. Dostupné z: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2701298/.

[18] DYER, Charles *R.* Machine Learning. In: [online] 2003. Dostupné z: http://pages.cs.wisc.edu/~dyer/cs540/notes/learning.html.

[19] SOKOLOVA, Marina a LAPALME, Guy. *A systematic analysis of performance measures for classification tasks*. ELSEVIER [online]. 2009. Dostupné z: http://rali.iro.umontreal.ca/rali/sites/default/files/publis/SokolovaLapalme-JIPM09.pdf.

[20] Confusion matrix. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation. Dostupné z: http://en.wikipedia.org/wiki/Confusion_matrix.

[21] SCHNEIDER, Jeff. 1997. Cross Validation. *Carnegie Mellon University: School of Computer Science* [online]. Dostupné z: http://www.cs.cmu.edu/~schneide/tut5/node42.html.

[22] Cross-validation (statistics). 2001-. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation. Dostupné z: http://en.wikipedia.org/wiki/Cross-validation_(statistics).

[23] ZELINKA, Ivan. *Biologicky inspirované výpočty: aneb vybrané statě z evolučních algoritmů* [online]. Dostupné z: http://arg.vsb.cz/data/Vyuka/BIV-AUI.pdf. Skripta. VŠB TU Ostrava.

[24] UCI DONALD BREN SCHOOL OF INFORMATICS & COMPUTER SCIENCE. *UCI Machine Learning Repository: Center for Machine Learning and Intelligent Systems* [online]. Dostupné z: https://archive.ics.uci.edu/ml/datasets.html.